Bug report for ENGF0002 CA 2.0 python

By Yadong Liu (Adam)

0. Basic game introduction

The game is programmed using tkinter. The purpose of this game is to eliminate buildings by drop bombs and land the plane. Plane will drop height by 40 pixel for every pass of row.

1. Issues

Bug 1: User can not drop bomb if previous bomb missed the building

Reason:

In "check_bomb" function, there's no statement to check if bomb has reached the ground. Hence, the bomb will drop forever if it is missed.

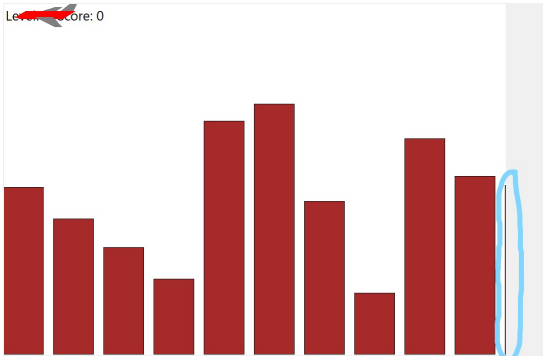
Fix:

Add a condition to check if the bomb reached the ground

```
    # make sure bomb does not go off the bottom of the screen
    if self.bomb.position.getY() >= CANVAS_HEIGHT:
        self.bomb.explode()
```

Bug 2: There is always a building block (like a line) at right edge of the screen

Issue pic:



Reason:

In "create_buildings", building_num is defined as 1200 // SPACING. However, our canvas has only width of 1000

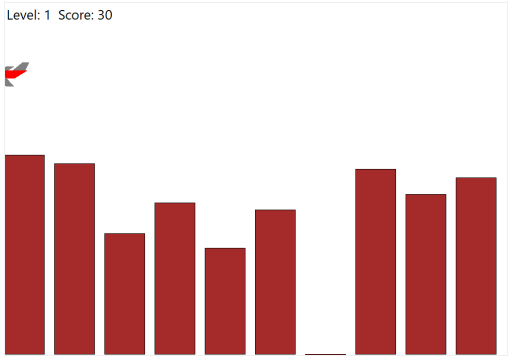
Fix:

```
    # change 1200 to CANVAS_WIDTH to make sure the program only generate     #
buildings that can display in canvas

    for building_num in range(0, CANVAS_WIDTH//SPACING):
        XXXXXXXXXXXXXXXXXXX
        XXXXXXXXXXXXXXXXXXX
```

Bug 3: The building still exists when height is <= 0

Issue pic:



Reason:

There is no cleanup for building height <=0, only shrink is called when bomb hitted the building

Fix:

- Way 1

```
    #Move cleanup method above shrink
    # add if statement in shrink
    if self.height <= 0:
        self.cleanup()
        #clean width, so building has no volume --collision to this
#building is disabled
        self.width = 0

    #in check_plane function change "==" to ">="
    if plane_body_bottom.getY() >= CANVAS_HEIGHT and
plane_body_bottom.getX() < 20:
        self.plane_landed()
```

- Way 2

```
    #ignore collision when plane Y coordinate is >= CANVAS_HEIGHT

    if plane_body_bottom.getY() >= CANVAS_HEIGHT:
        if plane_body_bottom.getX() < 20:
            self.plane_landed()
    else:
        for building in self.buildings:
            if (building.is_inside(plane_nose)
```

```
            or building.is_inside(plane_body_bottom)
            or building.is_inside(plane_wing)):
                self.game_over()
    # in "check_bomb()
    for building in self.buildings:
        if building.is_inside(self.bomb.position):
            self.bomb.explode()
            building.shrink()
            #check height
            if building.height <= 0:
                building.cleanup()
```

Bug 4: Game can't restart when it's running

Reason:

In "restart(self)" inside class "Display".

```
        self.canvas.delete(self.text)
```

However, text is not initialised until game is win or lose. Hence, when game is still running and no text displayed. This statement will cause error and therefore game can't be restarted.

Fix:

```
        #initial text as an empty string
        self.text = ""
```

Bug 5: It is impossible for User to hit the most right building:

Reason:

Because we only check speed per 10fps, hence the plane might not start outside canvas in next line.

Fix:

```
# Apply +100 to make sure plane starts outside canvas in order to drop bomb to
bottom-right buildding.
# In class "plane"

self.position.move(CANVAS_WIDTH + 100, 40)
```

2. Improvements

a. Plane Shifting

Problem and thoughts:

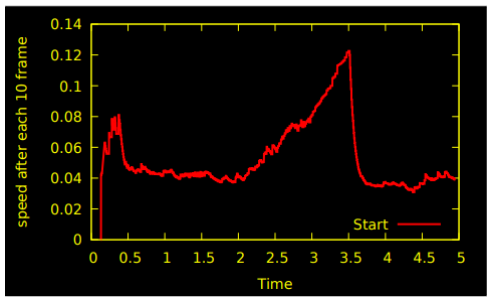
Plane is shifted when dropping a bomb.

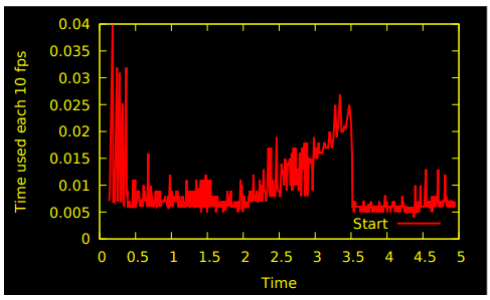
```
#Statement that defines movement of Plane
self.position.move(-4 * speed, 0)

#Statement that defines movement of Bomb
self.position.move(0, 8 * speed)
```

From code, we can find that the amount of movement depends on "speed". However, in "checkspeed" function, the programme only check speed per 10fps. Which causes problem when bomb is dropping (as there's extra statements to process)

I've plotted two graphs to show the relations between speed against time and time used for 10fps against time. (Sorry for just making config files from .gp used in python video 5)

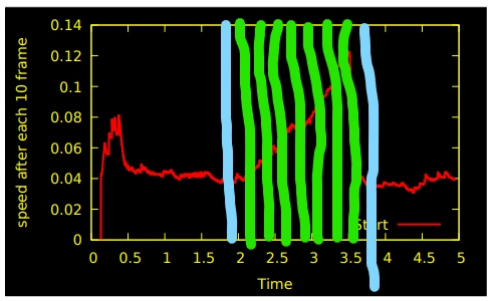




You could notice that there's a big increment and a sudden drop in speed between 2 and 3.5 seconds (which is between bomb drop and disappear). Personally, I think it is because of extra codes are processed each frame when a bomb is dropping. Hence after few loops of frames the time per 10fps increases and I guess the complexity is O(e^n). After bomb is exploded (not drawn), the process time dropped back to about the same level as before. Which proved that the time per 10fps is affected by Those extra amount of codes for bomb.

To Improve:

Back to graph, Blue line roughly represents the time interval for checking speed . Hence if we increase the rate to use EWMA equation, the difference between speed will become less. (Green lines)



We can simply increase the rate of checking framerates (however more resources used by CPU)

```
    def checkspeed(self):
        global speed
```

```
        self.framecount = self.framecount + 1
        time_passed = time() - start_time
        # Check for every frame
        if self.framecount == 1:
            now = time()
            elapsed = now - self.lastframe
            # speed will be 1.0 if we're achieving 60 fps
            if speed == 0:
                #initial speed value
                # At 60fps, 1 frames take 1/60 of a second.
                speed = 60 * elapsed
            else:
                # use an EWMA to damp speed changes and avoid excessive
#jitter
                speed = speed * 0.9 + 0.1 * 60 * elapsed
            self.lastframe = now
            self.framecount = 0
```

b. Bomb drops at constant speed

Problem and thoughts:

In reality, the velocity of bomb will affected by gravity. Hence it is necessary for us to make game more realistic

To Improve

```
    def __init__(self, canvas):
        XXXXXXXX
        XXXXXXXX
        #initial bomb drop speed
        self.dropsped = 0

    def move(self,):
        if self.falling:
            # v = u + at (each frame, hence t = 1)
            # one second is 60fps, each frame acc = 9.81/60 * speed
            # *speed is used to match "game world" speed rate
            self.dropspd = self.dropspd + 9.81/60 * speed
            self.position.move(0, self.dropspd * speed)

    def explode(self):
        self.falling = False
        # restore bomb speed
        self.dropspd = 0
```

c. No change made on difficulty between levels -- easy to get board

Problem and thoughts:

We could double the speed of plane to make game harder every time we passed a level

To Improve

```
    def __init__(self, canvas, x, y):
        XXXXXX
        XXXXXX
        #initial plane speed
        self.spd = -4

    def move(self):
        self.position.move(self.spd * speed, 0)
        XXXXXXXXX
        XXXXXXXXX
        XXXXXXXXX

    def next_level(self):
        #don't move to next level unless we've actually won!
        if self.won == False:
            return
        self.level = self.level + 1
        self.canvas.delete(self.text)
        self.canvas.delete(self.text2)
        self.plane.reset_position()

        #Double currnt speed when entering a new level
        self.plane.spd = 2 * self.plane.spd

        # buildings get narrower with each level
        self.building_width = self.building_width * 0.9
        self.create_buildings()
        self.won = False
        self.game_running = True
```

END