



Jan Wanot, 100878319

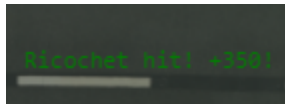
CS1830/CS1831 – 2017/2018 Individual Report

1. User manual

- **General information:**
 - Game can be run from the linked CodeSkulptor website.
 - In case the graphics are not loaded and errors appear, one should try running the game several times until all the images are cached.
 - Game was programmed with Python 3 in mind, but it also runs in Python 2 without any problems.
- **Basic controls:**
 - Movement: w forward, s backward, a strafe left, d strafe right.
 - Click in order to shoot, holding the mouse button fires automatically.
 - Press “E” or “Q” in order to toggle your weapon between a handgun and a rifle.
- **User interface:**



- **Enemies:**
 - *Simple bug:*  Walks towards the player at a slow but constant pace. Can be maneuvered to get stuck behind walls and obstacles. Dies in two hits, awards 100 points.
 - *Shielded bug:*  Similar to the simple bug, but is equipped with an energy shield on the front that deflects all bullets fired by the player. Can only be defeated if the player ricochets the bullet off a wall to hit its behind. Dies in just one successful hit and awards a whopping 300 points (plus ricochet bonus).



- *Bonus points:* Any enemy killed by a bullet that bounced off a wall, a tree, or another enemy first will award you an additional 350 points of ricochet bonus!

- **Pickups:**



- *Handgun ammo:* Gives player additional 50 bullets for the pistol.



- *Rifle ammo:* Gives player additional 250 bullets for the rifle.



- *Med kit:* Refills player health.

- **Trees:**



The trees inside the central area of the level are fully interactable. They both block the player and reflect bullets, as well as giving some shade to the monsters, allowing them to ambush and attack from hiding.

2. Development

The primary idea for the game that came up during our first meeting was by Tom, who wanted to make a simple twin stick/top down shooter game. It was during that first meeting we found the first graphics to use, as well as to assign everyone with responsibilities. We worked on that idea in the following weeks, culminating in the first prototype I made for the lab, which was just a simple level with placeholder background with working player-wall collision. We then refined it, adding new features and graphics(such as the enemies made by Johnathan, or the HUD made by Lantana), and then merged it with the menu and high score system that was worked on all this time by Luke.

While some of the graphics in the game are original and made by me (such as most of the level background, menu background and the modified shield enemy sprites), most of it are free game development assets from the internet. Here is a list of sources we have used:

<https://opengameart.org/content/lpc-beetle>

<https://opengameart.org/content/animated-top-down-survivor-player>

<https://yukikootomiye.deviantart.com/art/Sprites-Heart-Life-641296172>

<https://icons8.com/icon/15822/chevron-right>

<http://cgtextures.com/>

<https://www.turbosquid.com/3d-models/free-max-model-red-telephone-box/877648>

<https://free3d.com/3d-model/road-barrier-34045.html>

<https://opengameart.org/content/zombie-ui-pack>

<https://opengameart.org/content/trees-and-bushes>

Almost all the code we have written is original. However, we have borrowed several simple classes from our moodle labs and examples.

Vector class: http://py3.codeskulptor.org/#user301_dZJL5znAdk_0.py

Parts of the Sprite class: http://py3.codeskulptor.org/#user301_OUMtkJ0mk0ZuRcu.py

The Line class (and parts of the particle class for the bullets):

http://py3.codeskulptor.org/#user301_6s8FN8P1OH_0.py

3. Code

```
# Camera-related
viewDisplacement = (0, 0)

CAMERA_SPEED = 5

def cameraUp():
    global viewDisplacement
    if viewDisplacement[1]+CAMERA_SPEED <= 0 :
        viewDisplacement = (viewDisplacement[0], viewDisplacement[1] +
CAMERA_SPEED)

def cameraDown():
    global viewDisplacement
    if viewDisplacement[1]-CAMERA_SPEED >= -2143 :
        #print(viewDisplacement[1])
        viewDisplacement = (viewDisplacement[0], viewDisplacement[1] -
CAMERA_SPEED)

def cameraRight():
    global viewDisplacement
    if viewDisplacement[0]-CAMERA_SPEED >= -1283 :
        #print(viewDisplacement[0])
        viewDisplacement = (viewDisplacement[0] - CAMERA_SPEED,
viewDisplacement[1])

def cameraLeft():
    global viewDisplacement
    if viewDisplacement[0]+CAMERA_SPEED <= 0 :
        viewDisplacement = (viewDisplacement[0] + CAMERA_SPEED,
viewDisplacement[1])

def isInrange(position):
    return ((position[0]+viewDisplacement[0]) > 0) and
((position[0]+viewDisplacement[0]) < 960) and
((position[1]+viewDisplacement[1]) > 0) and
((position[1]+viewDisplacement[1]) < 740)
```

As I was tasked with level design in the group, one of my first challenges was how to allow for a game which playing field is bigger then the canvas on which we can draw. CodeSkuptor has no “camera” settings or options that we could adjust as far as I knew (maybe pygame does but I did not look into it since we were discouraged from using it), so I decided to just think about some other way in which this effect could have been achieved. I decided that the displacement of the “camera” from the right top corner of the screen would be stored as a tuple, and every time an object would be drawn, its position variable would be subtracted to give the adjusted position for drawing on screen, as follows:

```
    if self.adjusted:
        canvas.draw_image(self.image, centreSource, self.frameSize,
(self.pos.getP()[0]+Camera.viewDisplacement[0],
self.pos.getP()[1]+Camera.viewDisplacement[1]), self.frameSize,
self.rotation)
#(code taken from the sprite class)
```

This solved the problem for sprites, as when the camera is moved all of them have just the right relative position on the canvas compared to where they would normally be. The one thing left would be to ensure that the background would work just the same, which I had achieved by utilizing the parameters of the draw_image function intended for work with sprite sheets:

```
canvas.draw_image(BCKG, (CANVAS_CENTRE[0]-Camera.viewDisplacement[0],
CANVAS_CENTRE[1]-Camera.viewDisplacement[1]), CANVAS_DIMS, CANVAS_CENTRE,
CANVAS_DIMS)
```

I would then have the camera follow the player with a certain margin of freedom (adjustable with a constant):

```
CAMERA_SENSITIVITY = 0.4

def updateCamera():
    if (player.pos.x+Camera.viewDisplacement[0]) <
(CANVAS_DIMS[0]*CAMERA_SENSITIVITY):
        Camera.cameraLeft()
    elif (player.pos.x+Camera.viewDisplacement[0]) > (CANVAS_DIMS[0] -
(CANVAS_DIMS[0]*CAMERA_SENSITIVITY)):
        Camera.cameraRight()

    if (player.pos.y+Camera.viewDisplacement[1]) <
(CANVAS_DIMS[1]*CAMERA_SENSITIVITY):
        Camera.cameraUp()
    elif (player.pos.y+Camera.viewDisplacement[1]) > (CANVAS_DIMS[1] -
(CANVAS_DIMS[1]*CAMERA_SENSITIVITY)):
        Camera.cameraDown()
```

This setup works for the most part, but in some things require slight adjusting later on so that they work properly. For instance, while writing the fireBullet function much later I would have to account for this in order to get a proper direction to the cursor:

```
def fireBullet(self, position):
    if len(self.bullets) < 8 and not player.hasShot:
        self.bullets.add(Bullet(player.pos.copy(),
                                Vector(player.pos.x-
position[0]+Camera.viewDisplacement[0], player.pos.y-
position[1]+Camera.viewDisplacement[1]).normalize()*-20,
                                5,
                                1,
                                "yellow",
                                self.bullets))
```

Another part of the code that I am fond of is the level geometry and collisions. The visible background is just an image, but I prepared a set of coordinates for all the lines that make up for so that the collisions match it perfectly:

```
# Collision data for the level
MAPDATA = [((0,0), (0,2547)),
            ((0,2547), (448,2547)),
            ((448,2547), (448,2112)),
            ((448,2112), (576,1984)),
            ((576,1984), (1344,1984)),
            ((1344,1984), (1472,2112)),
            ((1472,2112), (1472,2547)),
            ((1472,2547), (1920,2547)),
            ((1920,2547), (1920,1408)),
            ((1920,1408), (1280,1408)),
            ((1280,1408), (1280,960)),
            ((1280,960), (1920,960)),
            ((1920,960), (1920,0)),
            ((1920,0), (1152,0)),
            ((1152,0), (1312,384)),
            ((1312,384), (960,608)),
            ((960,608), (608,384)),
            ((608,384), (768,0)),
            ((768,0), (0,0)),
            ((512, 1280), (576,1408)),
            ((576,1408), (448,1472)),
            ((448,1472), (384,1344)),
            ((384,1344), (512, 1280))]
```

This big array of tuples of tuples is then converted into a much nicer array of Line objects with the following code:

```
def makeLines(array):
    arrayOfLines = []
    for lines in array:
        arrayOfLines.append(Line(Vector(lines[0][0], lines[0][1]), Vector(lines[1][0],
        lines[1][1])))
    return arrayOfLines

lines = makeLines(MAPDATA)
```

This array is used for checking if a play, enemy or a bullet can move in a given direction. The player will be stopped and bullets are reflected along the normal of the wall in the proper manner, but an enemy will “slide” across the wall realistically in players direction as if looking for a way around, thanks to the following code:

```

def applyVelocity(self, vel):
    global lines
    vect = vel.copy().add(self.pos)
    for line in lines:
        if (line.distanceTo(vect) < line.thickness + self.radius and
            line.covers(vect)):
            vel -= line.normal * vel.dot(line.normal)
    self.pos.add(vel)

```

One feature that I came up with was when I realized that the middle “park” area of the map is very empty, and realized that by changing the order in which the draw functions of some objects are called in the draw handler I can have some objects be drawn on top of the others. From there, I got an idea to make a simple tree object that could be given a varied scale and rotation, and placed in specific spots to provide variety and added challenge (as it would hide enemy spawn points). I even made it so that each tree had a small collision-only “trunk” dependant on its scale that blocks player movement and reflects bullets realistically.

```

# Trees!

TREEMAGE =
simplegui.load_image('http://personal.rhul.ac.uk/zeac/123/game/tree.png')

class Tree:

    def __init__(self, pos, scale, rotation):
        self.pos = pos
        self.scale = scale
        self.radius = 5*scale
        self.rotation = rotation

    def vect(self):
        return Vector(self.pos[0], self.pos[1])

    def draw(self, canvas):
        canvas.draw_image(TREEMAGE,
                           (TREEMAGE.get_width()//2,
TREEMAGE.get_height()//2),
                           (TREEMAGE.get_width(), TREEMAGE.get_height()),
                           (self.pos[0]+Camera.viewDisplacement[0],self.pos[1]+Camera.viewDisplacement
[1]),
                           (TREEMAGE.get_width()*self.scale,
TREEMAGE.get_height()*self.scale),
                           self.rotation)

treesArray = [Tree((594, 923), 2, 45),
               Tree((1038, 921), 1.5, 123),
               Tree((965, 1140), 1.7, 78),
               Tree((670, 1210), 3, 56),
               Tree((320, 968), 1, 145),
               Tree((963, 1347), 1.75, 156)]

def checkForTree(tree, vect, radius):
    d = vect.copy().sub(tree.vect()).length()
    return d <= radius + tree.radius

```

The bullets themselves were also written by me as I handled collisions in the team, although Tom helped with the AllBullets class. I borrowed a lot of the vector operations and ideas on how to handle the physics from some of the moodle lessons, although I also figured out how to do a few things (like getting the normal vector from a bullet-tree collision by subtracting their positional vector and normalising them). Generally the bullet class itself is a heavily modified version of a Particle class from one of the moodle examples. The function for updating all the bullets was written completely by me though:

```
def updateBounces():
    for bullet in bullets.bullets:
        for line in lines:
            if (line.distanceTo(bullet.pos) < line.thickness +
bullet.radius and
            line.covers(bullet.pos)):
                if not bullet.hasBouncedBefore:
                    bullet.bounce(line.normal)
                    bullet.hasBouncedBefore = True
                else:
                    bullet.remove()
        for tree in treesArray:
            if checkForTree(tree, bullet.pos, bullet.radius):
                if not bullet.hasBouncedBefore:
                    bullet.bounce(tree.vect().sub(bullet.pos).normalize())
                    bullet.hasBouncedBefore = True
                else:
                    bullet.remove()
```

One piece of the code that I liked were the UpdateObjects and DrawObjects functions. At one point in code each object type (enemies, trees, bullets, items, etc.) would have its own update function that would be called in the update handler (tied to the timer, as all of our updates are tied to a timer and we could have easily implemented a pause feature if we didn't forget about it), and it worked the same way for the draw functions. At one point I realized it was a little tedious, so I just created a generalized object updating/drawing functions that could be drawn with any container of objects and would just update and draw each:

```
def updateObjects(iterator):
    if not isinstance(iterator, (list,)):
        #shallow copy to prevent set being modified during iteration
        temp = iterator.copy()
    else:
        temp = iterator
    for item in temp:
        item.update()

def drawObjects(canvas, iterator):
    if not isinstance(iterator, (list,)):
        #shallow copy to prevent set being modified during iteration
        temp = iterator.copy()
    else:
        temp = iterator
    for item in temp:
        item.draw(canvas)
```


The check there was due to the fact that the “dynamic” objects like enemies items and bullets used sets, while the “static” ones like trees and spawners would use arrays as they did not need to be removed or modified in any way. For safety I made it so that the functions would iterate on a shallow copy of a set, so that it would not be modified during iteration. This didn’t cause any problems in CodeSkulptor, but when I tried running my code on actual python on my computer it gave me errors so I wanted to play it safe.

I was also responsible for writing several other classes in the game, such as the enemy spawners, items and their spawners, text messages that pop up for two seconds when a ricochet is scored or item is picked up, and for implementing a rate of fire in the weapon class, as well as some other smaller tweaks, but the only other piece of code that I am particularly fond of are the two lines of code that are responsible for the “cutscene” when a new game starts:

```
Camera.viewDisplacement = (-650, 0)
player = Player(Vector(954, 1890))
```

4. Reflection

While I am overall happy with how the game had turned out to be, it does have a few critical shortcomings that I really wish we could have finished before we had submitted it. A lot of them are simply due to the fact that I retired and decided to do the OOP assignment during the last few hours before the deadline, when the rest of my team was struggling to get the game working in time and had problems with some of my code I could have helped them out with. This is why our game over screen does not appear when the player runs out of lives, for instance, forcing upon the player an honour system where he needs to press “o” after he runs out of lives to submit his high score.

Another, unrelated thing that I regret is not reading into the project specification enough in regards to some of the way the game was supposed to be coded. Aspects like abiding strictly to the OOP approach (which, as I found in many cases, is something not as obvious with Python as it is with Java), using vectors in place of tuples for coordinates and employing interaction classes for physics were all things that not only slipped past my attention, but were something that I had in many cases explicitly avoided and was quietly annoyed when my teammates employed them (thankfully I don’t recall ever actually ““correcting”” them in any case). Had I actually realized this, our code would probably be much better received and maybe it would even improve it in many cases.

One aspect I regret, not related to coding at all, was that one person who was registered to technically be in our group actually did not participate or show up even once. As a result, we have technically been a five-person group having to make game that would look good enough by a six-person group standards, which added additional pressure. But it is not really something that I feel like could have been foreseen or avoided in any way.

5. Course evaluation

I have had a very positive experience with this course. It introduced me to Python, which I expect to be quite useful for me in the future, as well as to some aspects of how video games work on technical level (which may also prove to be useful to me). In general, unlike a lot of the OOP labs and assignments we are doing from other subjects, the project allowed me to work more freely independently, and to find solutions to problems I set before myself on my own, which I thought was very motivating and quite pleasant as far as programming things goes. Since it was a lot bigger and unrestricted than other assignments I've had, it also showed the necessity to plan my code ahead and keep it maintainable in order not to get lost in it. The fact that it was a group project also learnt me a few things about working in a team as a programmer, keeping code styles consistent between team members, integrating modules with each other, and so on.

6. Suggestions

I feel like this course was pretty alright. If there was one thing I had a problem with it would have probably been the restrictions we had related to having to keep OOP approach at all points and things like that (using interaction classes, ect.), but I suppose these are justified since we need to learn good practice and so on. The only other suggestion I have is maybe to have the groups be assembled a little earlier and have them show off a prototype earlier as well, since it wasn't really until that point that we have started our work for real.