Introduction to Programming and Program Design Methods Final Project

Student Information

Name: Nathanael Setiawan

Student ID: 2440042866

Binusian ID: BN123727116

Binusian Email: nathanel.setiawan001@binus.ac.id

Phone Number: 081296030494

Class: B1BC

Lecturer Information

Name: Ida Bagus Kerthyayana

Lecturer ID: D5757

Table of contents:

Abstract:

The final project given to students of Program Design Methods and Introduction to Programming of BINUS International batch 2024 is a project where students are given creative liberty to create any application of sorts to serve a simple purpose. This project is designed for the student's individual self-learning and as a measurement of competency. The project that I made is a simple shooter simply title*"3D Shooter"*, which is a simple First Person Shooter game mostly utilizing the Ursina engine by Petter Amland.

The process of creating the final product is up to the individual developers, as long as they fully utilize the knowledge learnt from the courses they have initially took, as well as using previously untaught modules in their programs. *"3D Shooter"* Utilizes most of the basic concepts of Python, as well as utilizing a lot of the Ursina engine in terms of physics and rendering. The developers also are given creativity in what they want their final project to look like. I kept the mandatory requirements to be simple so the program could meet basic standards, but allowing for grander, unrealistic visions for the project.

The final product is a moderate success, accomplishing all the criteria set for myself, but not accomplishing any additional goals. The whole thing was a great learning experience as a lot of mistakes were made both inside the code as well as the project itself through mismanaged time and project mismanagement.

Introduction:

As a new BINUSian attending the international computer science course, one of the first courses the class of 2024 learnt is Introduction to Programming. In this course, students learn all about Python from its theoretical concepts all the way to the practical coding. As a way to test the student's abilities and understanding of Python so far, the lecturer, in this instance Mr.Ida Bagus Kerthyayana, gave the students freedom in creating something in Python with the concepts learnt thus far as well as enforcing further learning by requiring students to use some external libraries and modules that are not taught in the class. With this premise, I have decided I would like to create some kind of video game or at the very least a demo version of one.

The video game I have attempted to make is a simple First Person Shooter (FPS), utilizing a relatively recently developed (2019), open source game engine called the Ursina Engine (https://www.Ursinaengine.org/index.html) for Python. Created by, Petter Amland, this game engine is designed to help creating video games by really simplifying code and creating some prebuilt ready to use functions and classes for things such as the player character. Although the engine helps when creating a game, the documentation of Ursina Engine is not comprehensive for me and there are little to no tutorials in terms of systems I planned on creating, so I still had to test and tweak my code constantly to create some systems. And as this is the first time I've encountered the Ursina Engine as well as the first time of creating a big project, I was pretty excited to start.

The IDE I used during the project is Pycharm Community Edition 2020.2.1 and as this project's nature is open source, all progress will be uploaded to GitHub with this link: https://github.com/Nathanael126/3D-Shooter-PDM-Final-Project

Goals:

Project Purpose:

Primarily for the learning experience of the developer, it has the secondary purpose of entertaining any audience that fits within the nature of this project.

Targeted Audience:

The main audience of this project will be the lecturers; audience is not a big variable in the development of this project as the project nature serves as a learning tool rather for the developer rather than a practical application.

Project Aim:

The aim of this project is for the developer of the project to learn on how to create a new game through a library not taught directly to them, as well as demonstrate the competency by creating a functional experience for the user.

Objectives in-game:

Shooting the targets and gaining points is the main objective in-game.

Specification set by the developer:

-Functioning gun

-Functioning targets

-Functioning player

Additional vision (not mandatory) set by the developer:

-Multiple levels

-Multiple guns

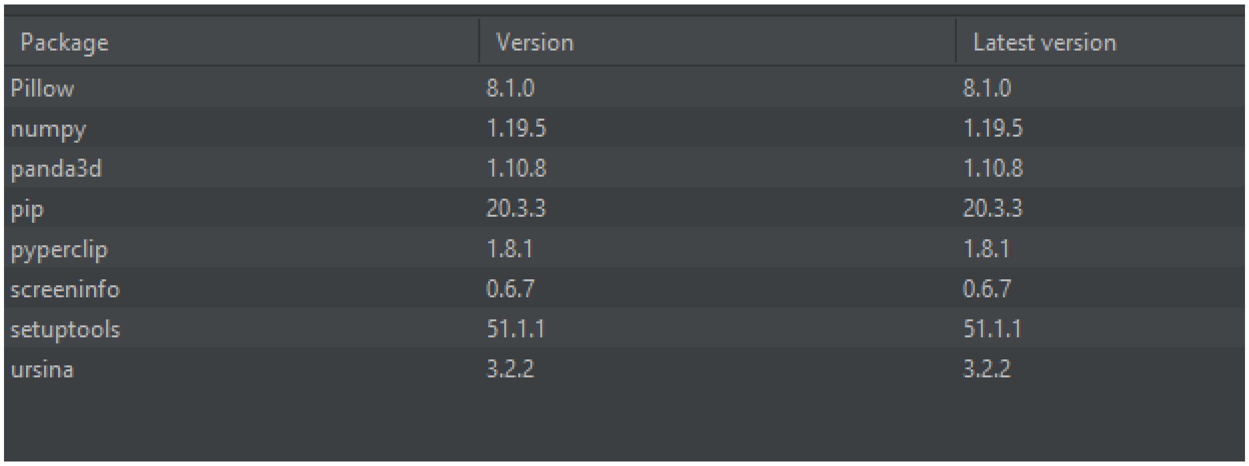
-AI

Methods:

Overview:

The game I created has been titled"*3D Shooter*". It is a simple shooting game demo which currently only has 1 level as a testing ground which is a shooting range. The game is written entirely in Python code, and mainly utilizes the Ursina Engine for most of its code.

All of the libraries present in the project files of *"3D Shooter"* are listed in the following screenshot:

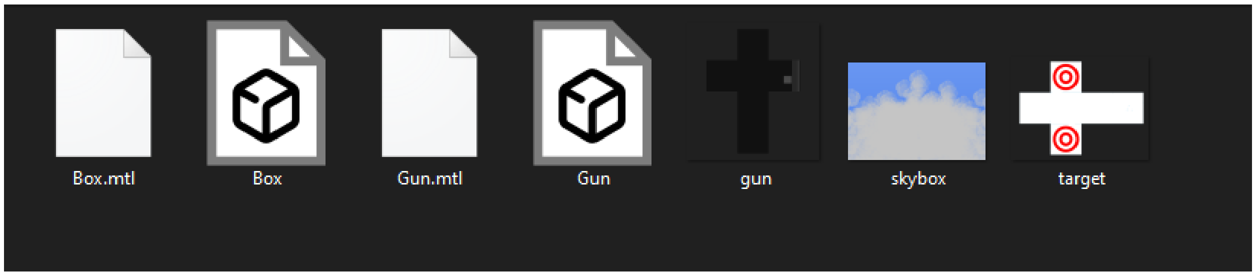


Package	Version	Latest version
Pillow	8.1.0	8.1.0
numpy	1.19.5	1.19.5
panda3d	1.10.8	1.10.8
pip	20.3.3	20.3.3
pyperclip	1.8.1	1.8.1
screeninfo	0.6.7	0.6.7
setuptools	51.1.1	51.1.1
ursina	3.2.2	3.2.2

The package utilized the most is the Ursina engine, most if not all the code utilizes functions and methods that are recognized by the Ursina engine. The engine is based on panda3d, so that package is installed by default. pip and setuptools are packages immediately included from the IDE Pycharm, and when first installing Ursina, Pycharm automatically downloaded Pillow, numpy, pyperclip, and screeninfo. Although I do not understand where they are utilized, as I did not reference them directly in my project, in the documentation these are all libraries that are considered dependencies for Ursina.

Assets:

All assets apart from the Ursina defaults are original and are included in the files. To ensure that the game works properly all files should be downloaded and the assets be placed in the right folder. Assets are shown below.

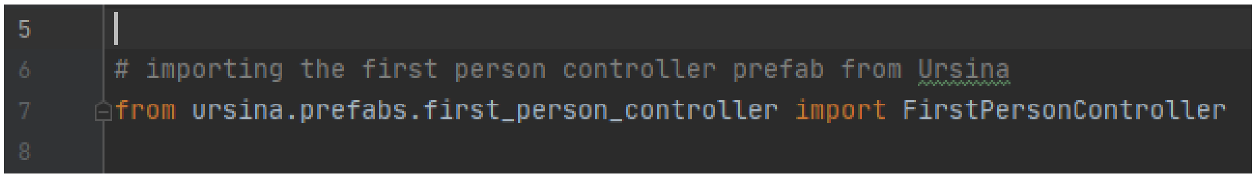


Prefab objects:

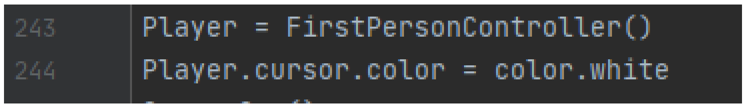
Prefab (prefabricated) are objects/methods that have been created and included in the Ursina engine, ready to be used immediately with very little code to create these objects.

First Person Controller:

The only prefab to be considered a prefab inside the documentation of Ursina itself, the prefab creates an object, which is assigned to the Player variable. The prefab makes the camera's parent to be the player object, and implements (WASD) movement controls as well as collision detection. The code needed to assign the player object to the variable is 2 lines long, the first line is for importing the prefab, and the second is assigning the variable, shown below.



```python

# importing the first person controller prefab from Ursina
from ursina.prefabs.first_person_controller import FirstPersonController

```



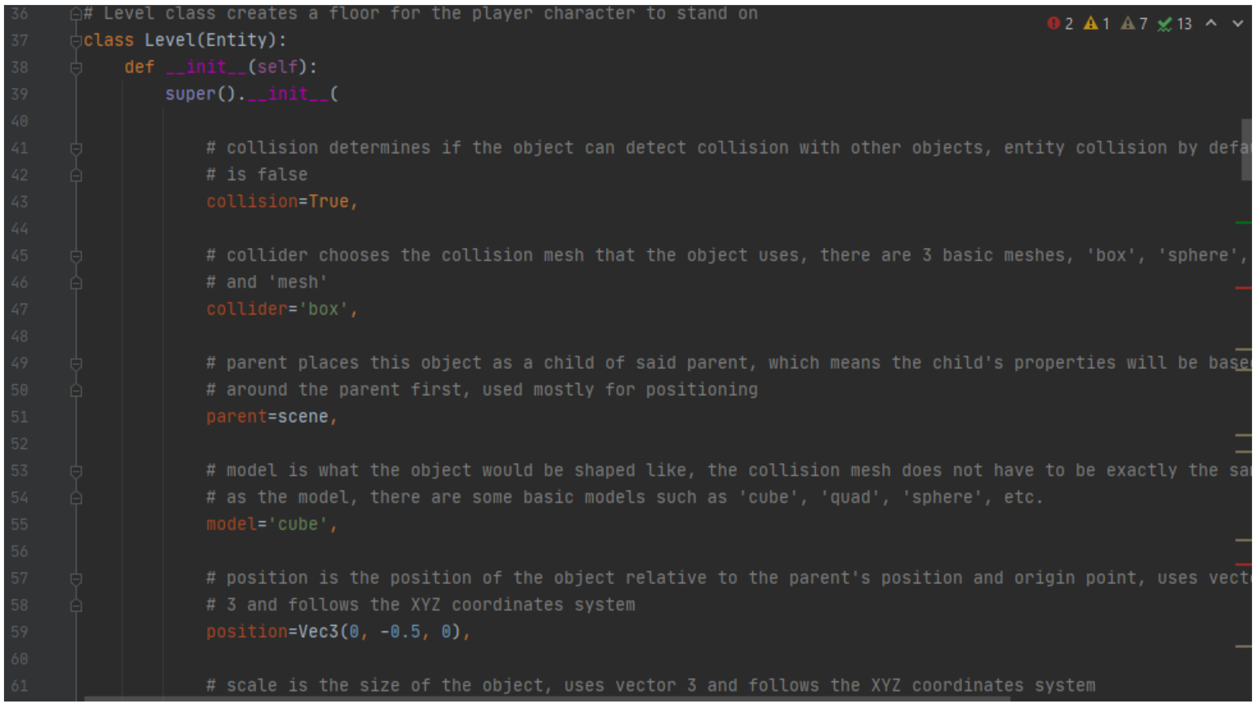
```python
Player = FirstPersonController()
Player.cursor.color = color.white
```

Classes:

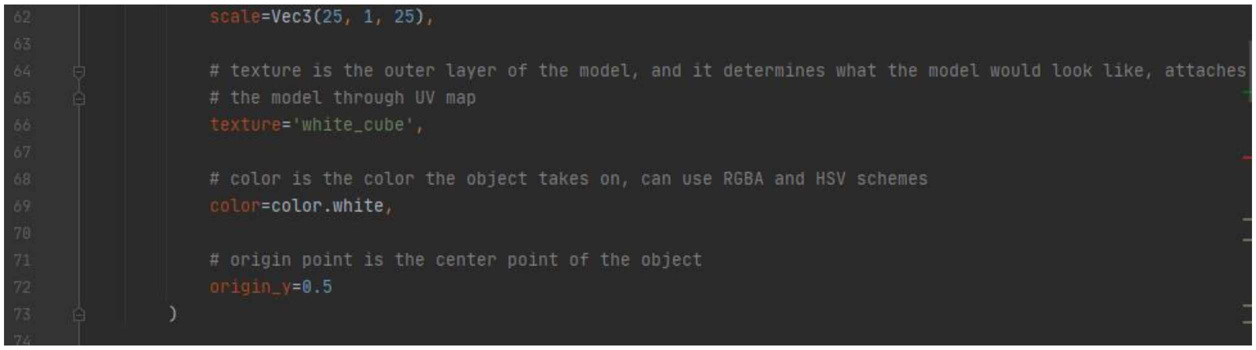
Classes are utilized a lot, as Python is an OOP language, naturally libraries made for it relies on OOP concepts. Ursina is no different, as all objects in-game are just instances of classes.

Level:

This class is used to just create a floor for the player to stand in; the code is shown below here.



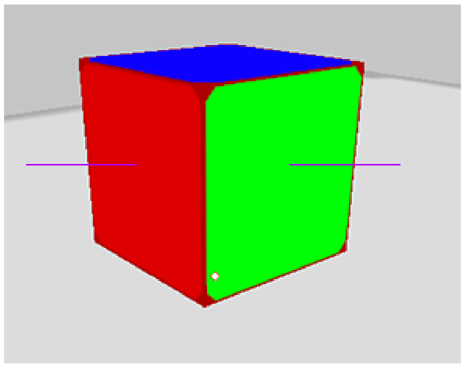
```python
# Level class creates a floor for the player character to stand on
class Level(Entity):
    def __init__(self):
        super().__init__(

            # collision determines if the object can detect collision with other objects, entity collision by defa
            # is false
            collision=True,

            # collider chooses the collision mesh that the object uses, there are 3 basic meshes, 'box', 'sphere',
            # and 'mesh'
            collider='box',

            # parent places this object as a child of said parent, which means the child's properties will be base
            # around the parent first, used mostly for positioning
            parent=scene,

            # model is what the object would be shaped like, the collision mesh does not have to be exactly the sa
            # as the model, there are some basic models such as 'cube', 'quad', 'sphere', etc.
            model='cube',

            # position is the position of the object relative to the parent's position and origin point, uses vect
            # 3 and follows the XYZ coordinates system
            position=Vec3(0, -0.5, 0),

            # scale is the size of the object, uses vector 3 and follows the XYZ coordinates system
```



```python
            scale=Vec3(25, 1, 25),

            # texture is the outer layer of the model, and it determines what the model would look like, attaches
            # the model through UV map
            texture='white_cube',

            # color is the color the object takes on, can use RGBA and HSV schemes
            color=color.white,

            # origin point is the center point of the object
            origin_y=0.5
        )
```

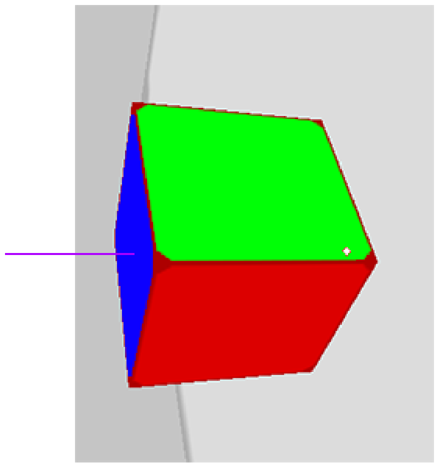
Target:

The target class is the class I spent the most time on. This is because the collision in Ursina is buggy, and there are little tutorials on how to use collision detectors such as raycast() and intersects(). The first collision detection method I tried was raycasting, but raycasts interact with their own models causing fake hits, and at the time, I did not comprehend how to know what entity it hits. The next method is intersects(), through a sample project I reverse engineered on how they work, but for some unknown reason, there is a bug where intersects() will not detect any collision from the X and Y axis of the Target class. Below is a visual on what the bug looks like.



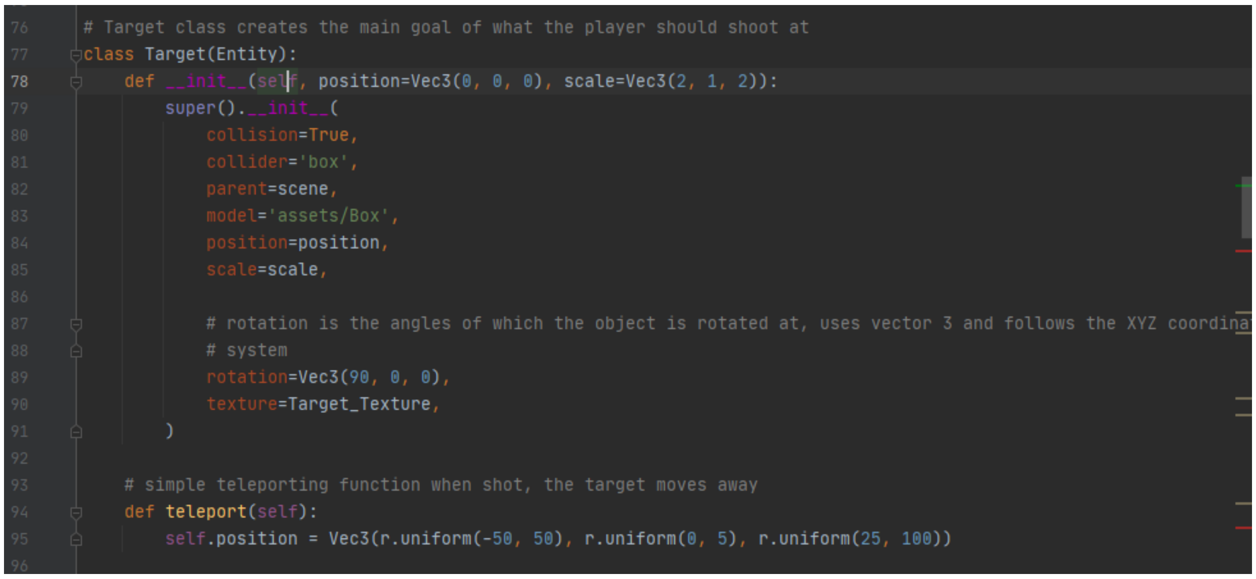
Blue = Y Axis
Green = Z Axis
Red = X Axis
Purple = Bullet path

If the bullet passes through either the Z or X axes, the bullet will not register any collisions



But it will register collision if the bullet hits the Y axis, hence only needing a 90 degree turn in either X or Z axes

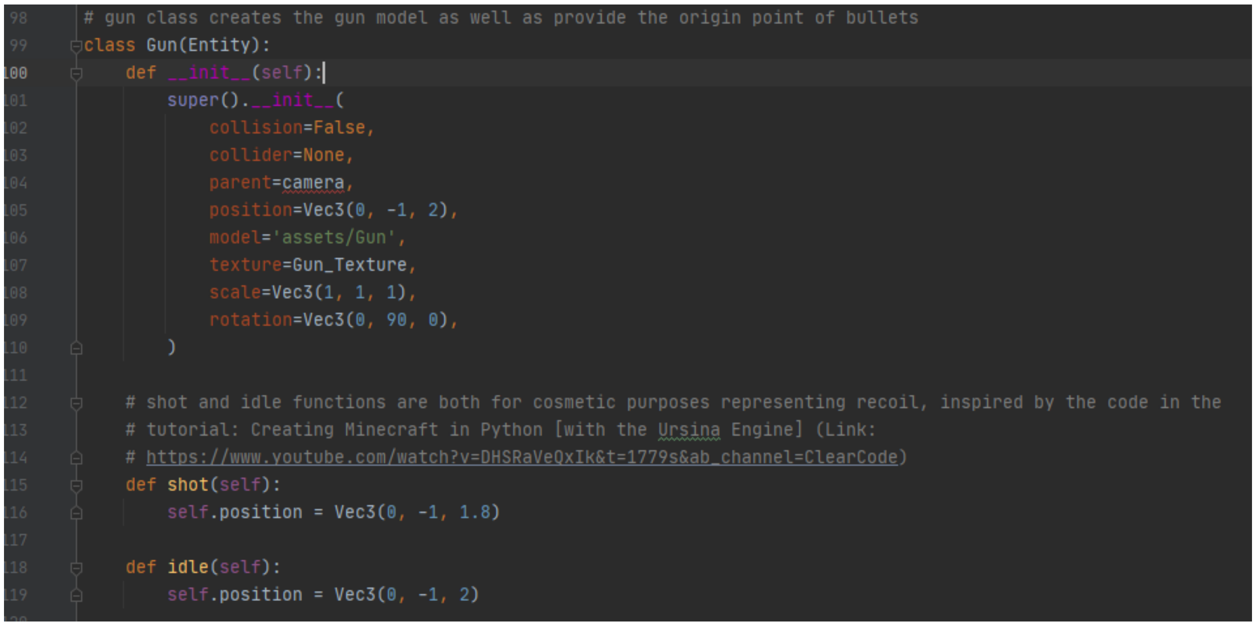
This bug cause a lot of issues for me, and with inconsistent collision detection from the target class as well as just learning how hit detection works through the API documentation, it took a significant portion of time, and the target is reduced to become a basic cuboid object, shown in the code below.



```
# Target class creates the main goal of what the player should shoot at
class Target(Entity):
    def __init__(self, position=Vec3(0, 0, 0), scale=Vec3(2, 1, 2)):
        super().__init__(
            collision=True,
            collider='box',
            parent=scene,
            model='assets/Box',
            position=position,
            scale=scale,

            # rotation is the angles of which the object is rotated at, uses vector 3 and follows the XYZ coordina
            # system
            rotation=Vec3(90, 0, 0),
            texture=Target_Texture,
        )

    # simple teleporting function when shot, the target moves away
    def teleport(self):
        self.position = Vec3(r.uniform(-50, 50), r.uniform(0, 5), r.uniform(25, 100))
```

Gun:

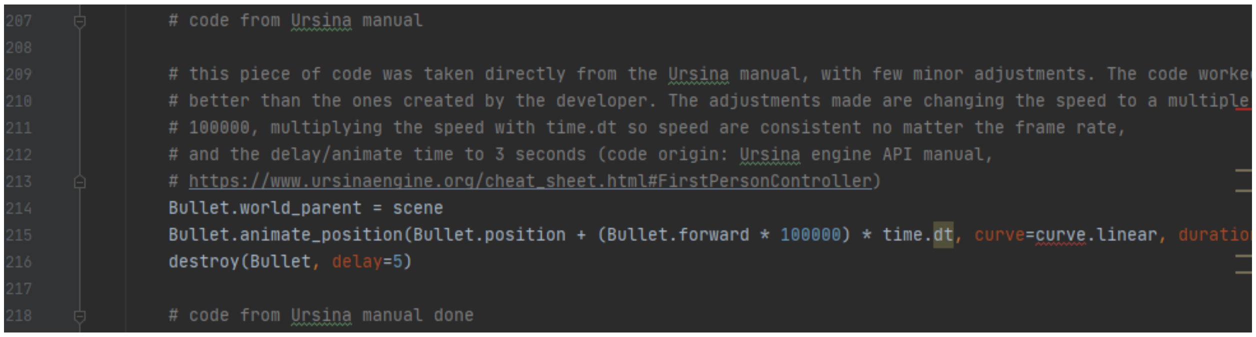
The gun is just a stretched sphere model with no collisions, meant to act as a visual of a gun and the origin point of bullets. The parent of the gun is the camera so it will follow the camera no matter what. The shot and idle functions are inspired by Clear Code's YouTube tutorial, credits at the bottom of the document, Code is shown below.



```python
# gun class creates the gun model as well as provide the origin point of bullets
class Gun(Entity):
    def __init__(self):
        super().__init__(
            collision=False,
            collider=None,
            parent=camera,
            position=Vec3(0, -1, 2),
            model='assets/Gun',
            texture=Gun_Texture,
            scale=Vec3(1, 1, 1),
            rotation=Vec3(0, 90, 0),
        )

    # shot and idle functions are both for cosmetic purposes representing recoil, inspired by the code in the
    # tutorial: Creating Minecraft in Python [with the Ursina Engine] (Link:
    # https://www.youtube.com/watch?v=DHSRaVeQxIk&t=1779s&ab_channel=ClearCode)
    def shot(self):
        self.position = Vec3(0, -1, 1.8)

    def idle(self):
        self.position = Vec3(0, -1, 2)
```

Bullet:

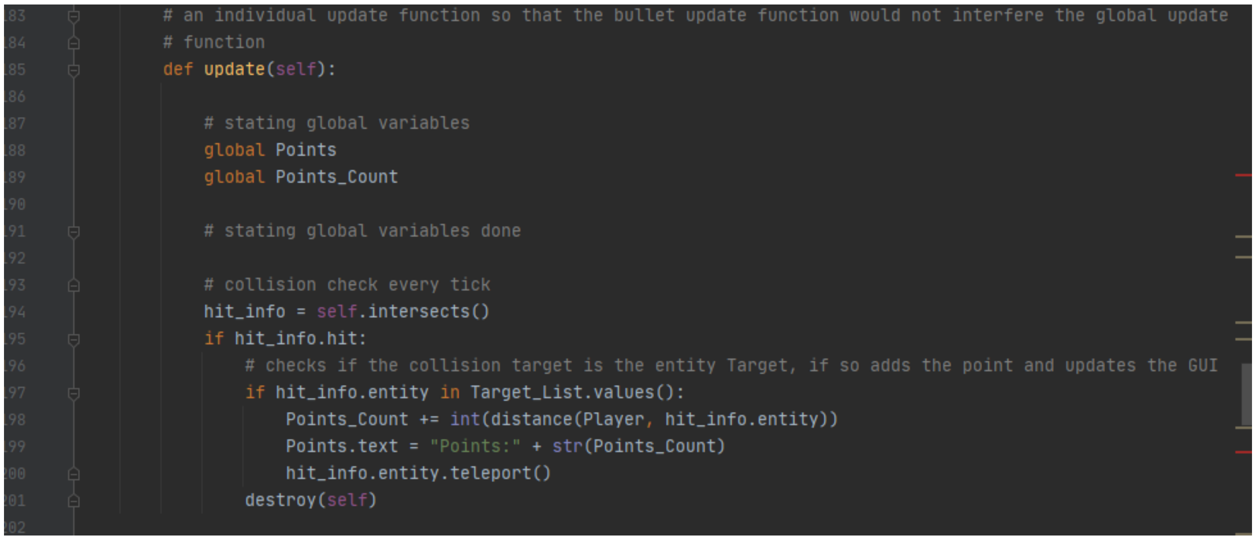
The bullet is a small cuboid with collision detection. The primary action of a bullet contains codes which are unoriginal with minor changes. This is because I could not find any other feasible way of creating the bullet behavior going forward while getting shot out of the gun. Credits at the bottom of the document, the code in particular is shown below as well as the original source code.



```python
        # code from Ursina manual

        # this piece of code was taken directly from the Ursina manual, with few minor adjustments. The code worke
        # better than the ones created by the developer. The adjustments made are changing the speed to a multiple
        # 100000, multiplying the speed with time.dt so speed are consistent no matter the frame rate,
        # and the delay/animate time to 3 seconds (code origin: Ursina engine API manual,
        # https://www.ursinaengine.org/cheat_sheet.html#FirstPersonController)
        Bullet.world_parent = scene
        Bullet.animate_position(Bullet.position + (Bullet.forward * 100000) * time.dt, curve=curve.linear, duratio
        destroy(Bullet, delay=5)

        # code from Ursina manual done
```

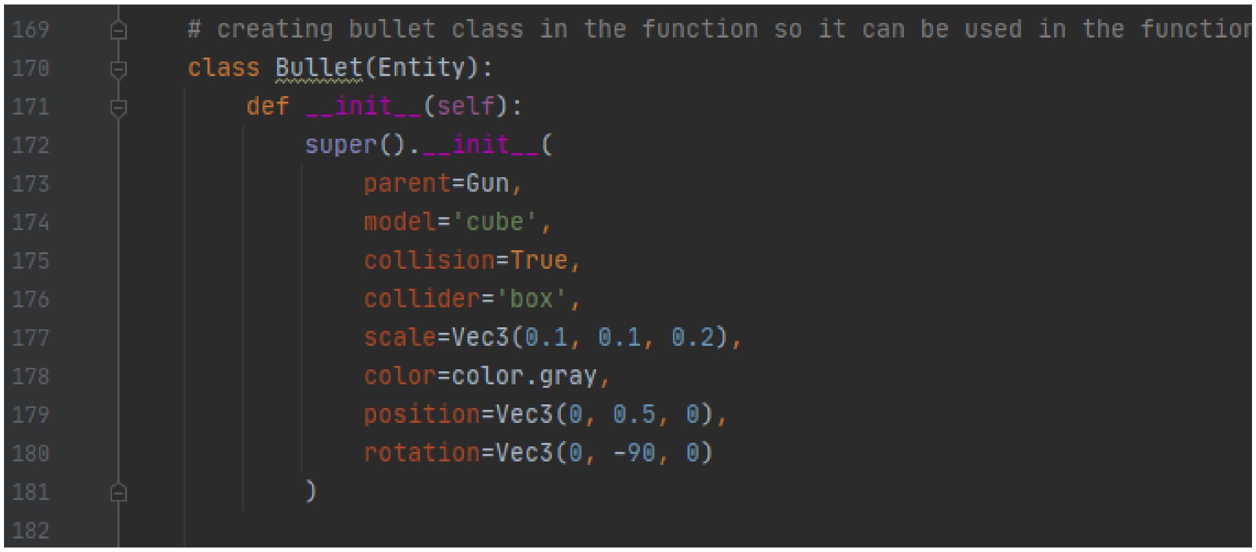
```python
        bullet.world_parent = scene
        bullet.animate_position(bullet.position+(bullet.forward*50), curve=curve.linear, 
duration=1)
        destroy(bullet, delay=1)
```

The collision detection, just like the target, was difficult to program in, but not as difficult as the target, the bullet has better collision detection and because it always moves, there is always a higher chance of any collision happening, the collision and point calculation code can be seen below.



```
    # an individual update function so that the bullet update function would not interfere the global update
    # function
    def update(self):

        # stating global variables
        global Points
        global Points_Count

        # stating global variables done

        # collision check every tick
        hit_info = self.intersects()
        if hit_info.hit:
            # checks if the collision target is the entity Target, if so adds the point and updates the GUI
            if hit_info.entity in Target_List.values():
                Points_Count += int(distance(Player, hit_info.entity))
                Points.text = "Points:" + str(Points_Count)
                hit_info.entity.teleport()
            destroy(self)
```

The rest of the class is shown below.

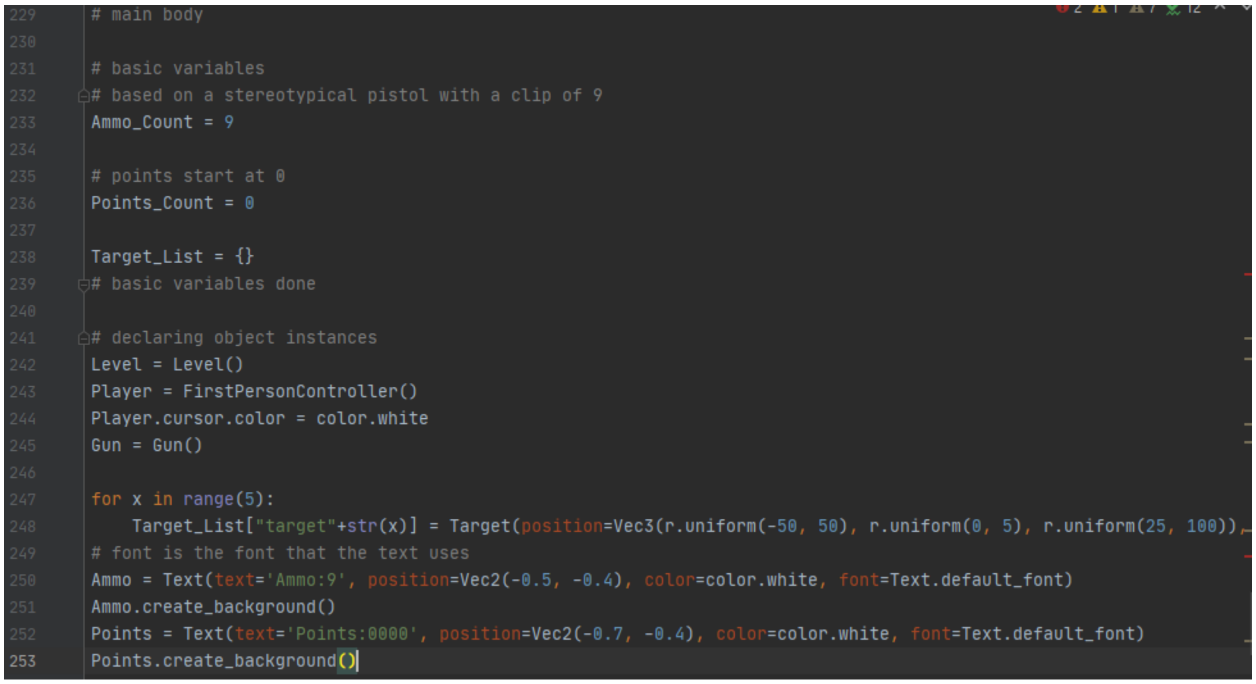


```
# creating bullet class in the function so it can be used in the functior
class Bullet(Entity):
    def __init__(self):
        super().__init__(
            parent=Gun,
            model='cube',
            collision=True,
            collider='box',
            scale=Vec3(0.1, 0.1, 0.2),
            color=color.gray,
            position=Vec3(0, 0.5, 0),
            rotation=Vec3(0, -90, 0)
        )
```

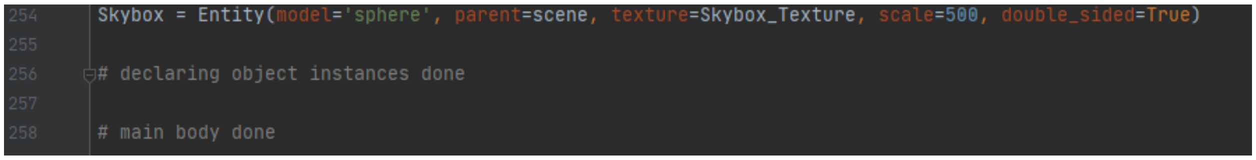
Miscellaneous:

Main body:

The main body of the class is used to create instances of the many different classes, and to just generally run the game as well as creating smaller things that do not really need a class such as a skybox which does not interact much. The code is shown below.



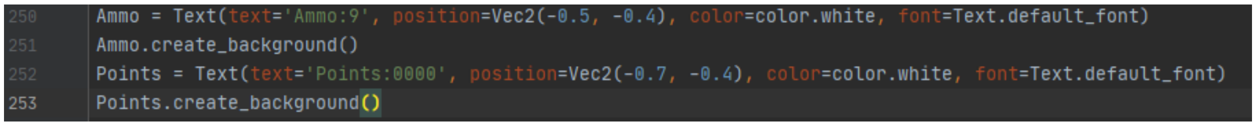
```
# main body

# basic variables
# based on a stereotypical pistol with a clip of 9
Ammo_Count = 9

# points start at 0
Points_Count = 0

Target_List = {}
# basic variables done

# declaring object instances
Level = Level()
Player = FirstPersonController()
Player.cursor.color = color.white
Gun = Gun()

for x in range(5):
    Target_List["target"+str(x)] = Target(position=Vec3(r.uniform(-50, 50), r.uniform(0, 5), r.uniform(25, 100)),
# font is the font that the text uses
Ammo = Text(text='Ammo:9', position=Vec2(-0.5, -0.4), color=color.white, font=Text.default_font)
Ammo.create_background()
Points = Text(text='Points:0000', position=Vec2(-0.7, -0.4), color=color.white, font=Text.default_font)
Points.create_background()
```



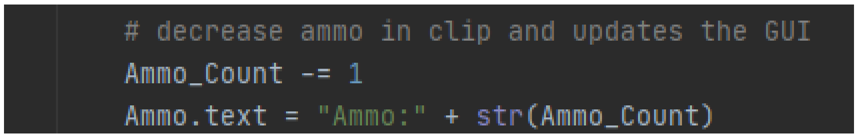
```
Skybox = Entity(model='sphere', parent=scene, texture=Skybox_Texture, scale=500, double_sided=True)

# declaring object instances done

# main body done
```

GUI:

The GUI is the final visual aspect of the game; it is created in the main body and it is updated when the variables change for that GUI. Code is shown below.

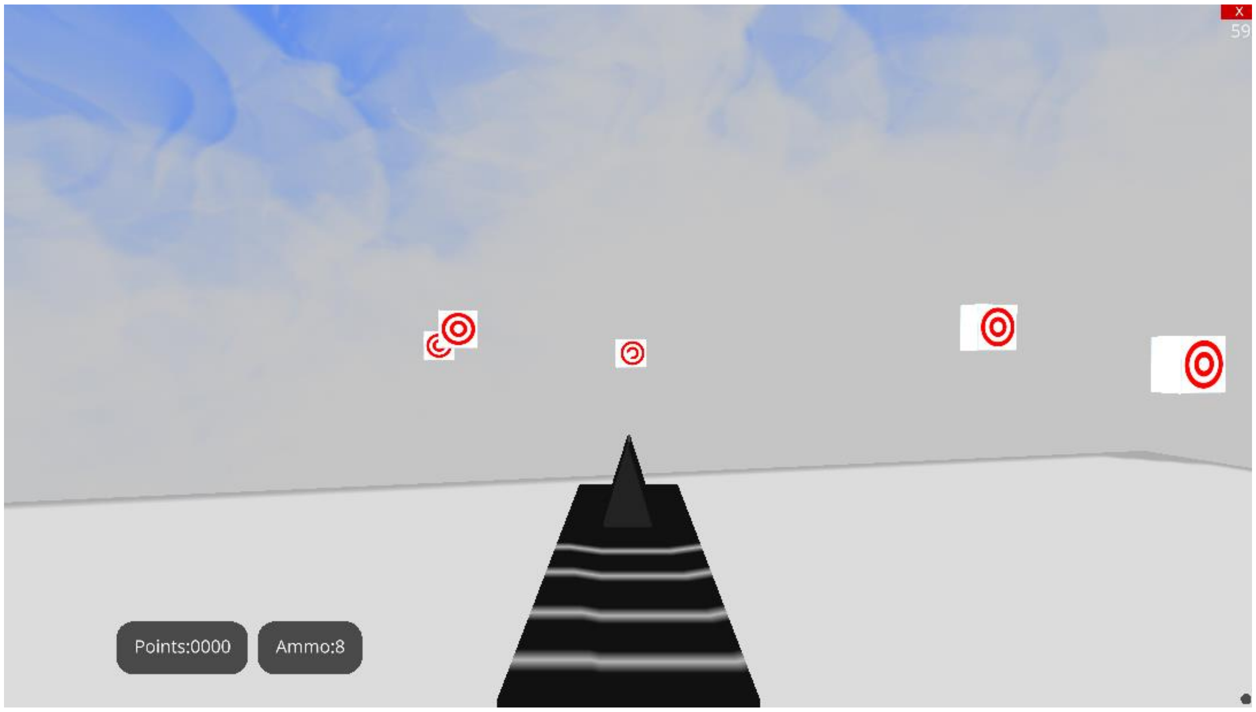


```
Ammo = Text(text='Ammo:9', position=Vec2(-0.5, -0.4), color=color.white, font=Text.default_font)
Ammo.create_background()
Points = Text(text='Points:0000', position=Vec2(-0.7, -0.4), color=color.white, font=Text.default_font)
Points.create_background()
```



```
# decrease ammo in clip and updates the GUI
Ammo_Count -= 1
Ammo.text = "Ammo:" + str(Ammo_Count)
```

Results:



The mandatory requirements are already achieved, but the extra vision is not. The gun, targets, and player all functions properly, and even have features such as teleporting targets and a bit of visual knockback. Unfortunately due to poor time management and planning the bigger scope and vision of the game is unfortunately not possible in the given timeframe. That said, most concepts of OOP are utilized in this project, and it was a great learning tool for me.

Conclusion:

The main goals and purpose of the project has been met and it did teach me, the developer quite a lot in terms of how to utilize the Ursina engine. There was a lot of challenging aspects in learning an engine which is not well known yet, and I had to test each function individually to see both how they work and to make sure that there are no bugs in the final version of the game.

But because of my own slow process of learning, the need for constant experimentation and play testing, and little tutorials or vague documentation of the Ursina engine, the scope of the game is very small and the final product is a very small experience. The final version of the game came up very differently from what I had envisioned at the very beginning, and compared to other projects of a similar nature, it will look unimpressive.

Despite the negatives, *“3D Shooter”* met all requirements which I have specified, and it already utilized a lot of concepts from Python, OOP, and Ursina engine. And because of these reasons, the project *“3D Shooter”* can be considered a moderate success.

Reflection:

This project definitely taught me a lot, not only about Python, but also in aspects of time and project management. In terms of Python, it was great practice for basic Python coding, as well as a great learning experience on how to tackle an unfamiliar engine or module, in this case the Ursina engine. The engine, although designed to be a beginner friendly engine since it easily integrates with Python, was difficult for me to learn. My inexperience really affected the flow of the project.

Another thing that I was inexperienced was project and time management. Initially, I underestimated how much I had to learn with the engine, so I took some more time brainstorming on what the final project should look like instead of getting familiar with the engine. This lead to me having to learn on the spot, and experimenting while coding the game at the same time. This alongside poor time management really hurts the final product. I have learnt from my mistakes and will attempt to improve my skills not only in coding, but in time management and project management skills.

Contributions:

As previously stated in this document, the Ursina engine is very limited in terms of the amount of content available in the forms of tutorials. Here will be a useful reference point for using the Ursina engine for any future purposes.

Collision:

Collision, as previously stated, behaves strangely inside of the Ursina engine. From thorough testing of my bullet and target, I have concluded that the cuboid collisions using intersects() does not work on the X and Y axes faces. This can be improved by rotating the cuboid 90 degrees so that the collisions can occur. However, this still does not allow full cuboid coverage. It is because of this it is recommended to use raycast() and boxcast() classes. Raycast is much more reliable because it is able to detect collisions much more reliably than intersects(). Unfortunately this is very sensitive if an event should occur if the ray touched anything, but it is much more reliable in terms of the bullet touched any specific entity. Another property all these collision detection should utilize is the hit information, as this object instance gives the information of what is hit. Below are some useful lines that should help in collision detection.

```
#original properties of the raycast class according to the API manual

Ray = raycast(origin, direction=(0,0,1), distance=inf, traverse_target=scene, ignore=list(), debug=False)

#example of initializing this class

Ray = raycast( Spawner,
direction(0,90,0),distance=10,traverse_target=scene,ignore=list(wall),debug=False)

#entity detection example

If Ray.entity = Block1: print(“Hello!”)

#original properties of intersects property according to the API manual

intersects(traverse_target=scene, ignore=(), debug=False)

#example of using this property

Intersection = self.intersects(traverse_target=scene, ignore=(wall), debug=False)

#entity detection example

If Intersection.entity = Block1: print(“Hello!”)
```

Animate movement:

One of the ways that movement is done inside of the Ursina engine is through animating that movement. This is much more efficient in terms of code length as it only requires 1 line instead of using an alternative like putting movement inside of the update function which requires the objects XYZ position as well as the targeted XYZ position that the object will travel towards. Animating movement using animate_position() property. Examples are shown below.

```
#original properties of animate_position property according to the API manual

 animate_position(value, duration=.1, **kwargs)

#example of using this property

self.animate_position(self.position+(self.forward*100*time.dt), duration=5)
```

Credits, sources, and citations:

Images:

Binus University International: https://international.binus.ac.id/

Learning links:

Clear Code, 2, December, 2020, *Creating Minecraft in Python [with the Ursina Engine], https://www.youtube.com/watch?v=DHSRaVeQxIk&ab_channel=ClearCode*

Petter Amland, *Ursina Engine documentation, https://www.Ursinaengine.org/documentation.html*

Petter Amland, *Ursina Engine cheat sheet, https://www.Ursinaengine.org/cheat_sheet.html*

Code origins:

Clear Code, 2, December, 2020, *Creating Minecraft in Python [with the Ursina Engine], https://www.youtube.com/watch?v=DHSRaVeQxIk&ab_channel=ClearCode*

Petter Amland, *Ursina Engine cheat sheet First Person Controller, https://www.Ursinaengine.org/cheat_sheet.html#FirstPersonController*

Libraries:

Petter Amland, *Ursina Engine, https://www.Ursinaengine.org/*

3, Jan, 2021, *Pillow, https://pypi.org/project/Pillow/*

2020, *Numpy, https://numpy.org/*

2021, *panda3d, https://www.panda3d.org/*

15, Dec, 2020, *pip, https://pypi.org/project/pip/*

11, Oct, 2020, *pyperclip, https://pypi.org/project/pyperclip/*

28, Dec, 2020, *screeninfo, https://pypi.org/project/screeninfo/*

9, Jan, 2021, *setuptools, https://pypi.org/project/setuptools/*