Games Individual Report (CS1822)

Erikas Vieraitis

Erikas Vieraitis

Dungeon Hero

(Dungeon Crawler Shooter Game)

This Report is made for the purpose of describing the game our group has created in all of its different general and technical aspects, as well as what certain decisions and paths we took to achieve the criteria needed for this game. The report will have 2 main sections; 1) An overview containing a description of the game, how we designed it, approached tasks and created it, 2) A technical section analysing the code and the steps we took to implementing it.

Overview

Description

So the game begins with the player on the main menu which introduces the title of the game and the main premise, as well as controls. You play as a human who's been sent down to the infinite dungeon, with the goal to see how far you can get. Along your journey you will encounter randomised enemies which you will have to kill with guns. There are different varieties of enemies, from skeletons that shoot to zombies that chase the player. Items can drop from enemies including ammo, health and weapons. Once you kill all enemies in the room, you go to the next one where it continually gets harder and harder each room with increased enemies as well as their health and speed. The goal of this game is to see how far one can get in the infinite dungeon

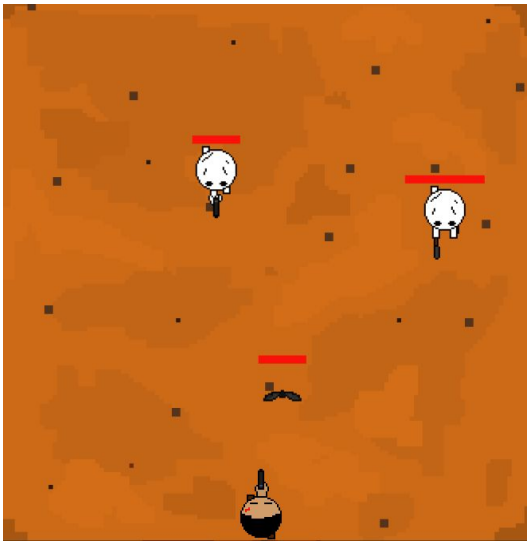
I will describe a **user manual** explaining what you have to do as well as all of the features in the game. The game is keyboard controlled using WASD to move, and arrow keys to shoot. When an enemy drops an item the player can pick up the item with E. The player can change weapons with the numbers 1 to 4 depending on the weapon slots.

There are 5 different enemies you will encounter in the game. The skeleton with a knife runs towards the player to hit him. Skeleton with a gun shoots the player from a distance. Zombies walk towards the player slowly and the Bat and Spider go towards the player quickly. There are 5 different guns enemies can drop that can be equipped. The Starting Pistol, Mac11, Ak47, Shotgun and Desert Eagle, each with varying ammo, fire rate and damage. Other than weapons, enemies can drop a small ammo which recharges the gun they are holding, an ammo

crate which recharges all of the guns they have, or a heart which restores one point of health. Each time an enemy is killed the player is rewarded with points. When all are killed in the room, the player can traverse to the top of the room to enter the next where one extra enemy will be added (Max 10 in one room) as well as every 5 rooms their health and speed will increase. When the player is either hit by an enemy or an enemy bullet then they have 1 second invincibility before they can take damage again, and once the player's health is 0, then a game over screen appears stating their points and their furthest room they reached.

To play the game go to this link:
https://py3.codeskulptor.org/#user305_selMoQMe15_6.py

The Design and Creation Process of the game



When given the task of making the game, we were given a list of required specifications that we had to achieve within it. The main ones were to create a game that used SimpleGui, to use a premade Vector class to control coordinate manipulation in the game and to have a form of collision within our game. We pursued the idea of a top down shooter and thought that would be perfect as projectiles use collisions and the vector class too. Originally the game was going to have bosses and an ending, but for a game of this style, we thought replayability was important so we decided to make it endless. Next time, I think I would program in bosses every few rooms for the challenge but still keep it endless to maintain the games structure.

We used an online coding website called CodeSkulptor which allowed us to code online and share the code easily with just a simple link so any of us could access it and add to each other's code. This is part of the reason we all worked well together, as the group dynamic was near perfect and it made it really easy to be able to just crack on with the project without even having to be in the same room. The most important aspect of the entire design and creation process was SimpleGui. SimpleGui is mainly a graphical python unit which allows the creation of GUI's and graphical environments. It also contained mouse and keyboard handlers which are useful when creating video games. We first used it to draw the dungeon and the character and then incorporated movement for the character around the canvas and then enemies and projectiles

as well as items, interacting together on the same screen all using SimpleGui. As well as the actual game, the Menu Screen and GameOver screen were created with it, so it can be used for an endless variety of interfaces.

I tasked myself with 2 main jobs, creating the sprites and animations, as well as being the main structural programmer. The easy part of my job was the sprites as they just required drawing some sprite sheets using the online sprite creator Piskel. For the programming side I would lay out all of the classes as well as programming each essential method the classes would use. I also programmed a lot of the game logic for the player and each of the items/weapons as well as the players movement and shooting which used the vector class, the interactions with the environment, animations, and the UI with a main menu and an end screen.

We wanted to use Super Constructors where the classes of the player and enemies would use the parent class “Character” but I found it quite tricky to wrap my head around for the first time. But this way ensured more easily readable code and more efficient code in the long run so I persevered. One of the other harder tasks I had was to program the interaction of each weapon and the player's inventory as it required the logic to detect a weapon swap when the inventory was full. Some of the problems encountered was sometimes the right weapon wouldn't appear in the players inventory when full and the weapons would be drawn wrongly, but with some help from my group we managed to get it fully working.

Ben and Haydn had many tasks but their biggest one was using the vector class to be able to get the enemies to walk and shoot in the player's direction. Using solely the vector class for coordinate manipulation proved very useful as we could use it for everything, from the player/enemy movement, to each of the projectiles and items in the game. The class had many pre-build methods which allowed for a lot of the functions to be programmed in easily. Valerie did most of the coding for the collisions which detected if an enemy was touching something. If so then they would bounce off slightly so it’s not stuck to the entity. Projectiles used collisions too to detect damage to either player or enemy. The game that we created met most of our initial criteria when we first decided what it was going to be. The reason the game came together so well was because of our group dynamic. We all got along very well and worked very well together too, with clear

communication and an outline of what each person had to do to contribute based off of their skill level in programming.

Finally to conclude the overview, I think this coursework and our game was a success as we created a very fun and interactive experience for the user with unlimited replayability. To improve the game's general feel and look I think we could have had smoother movements for each entity so it feels less static, but I also think the group could have met more to bounce ideas off face to face as many things could have been designed in a better way with some real life discussion. Overall, this game is well-crafted; it requires some skill and getting used to, but it meets the ultimate requirement of all games which is to be fun and enjoyable.

Erikas Vieraitis

Technical Analysis

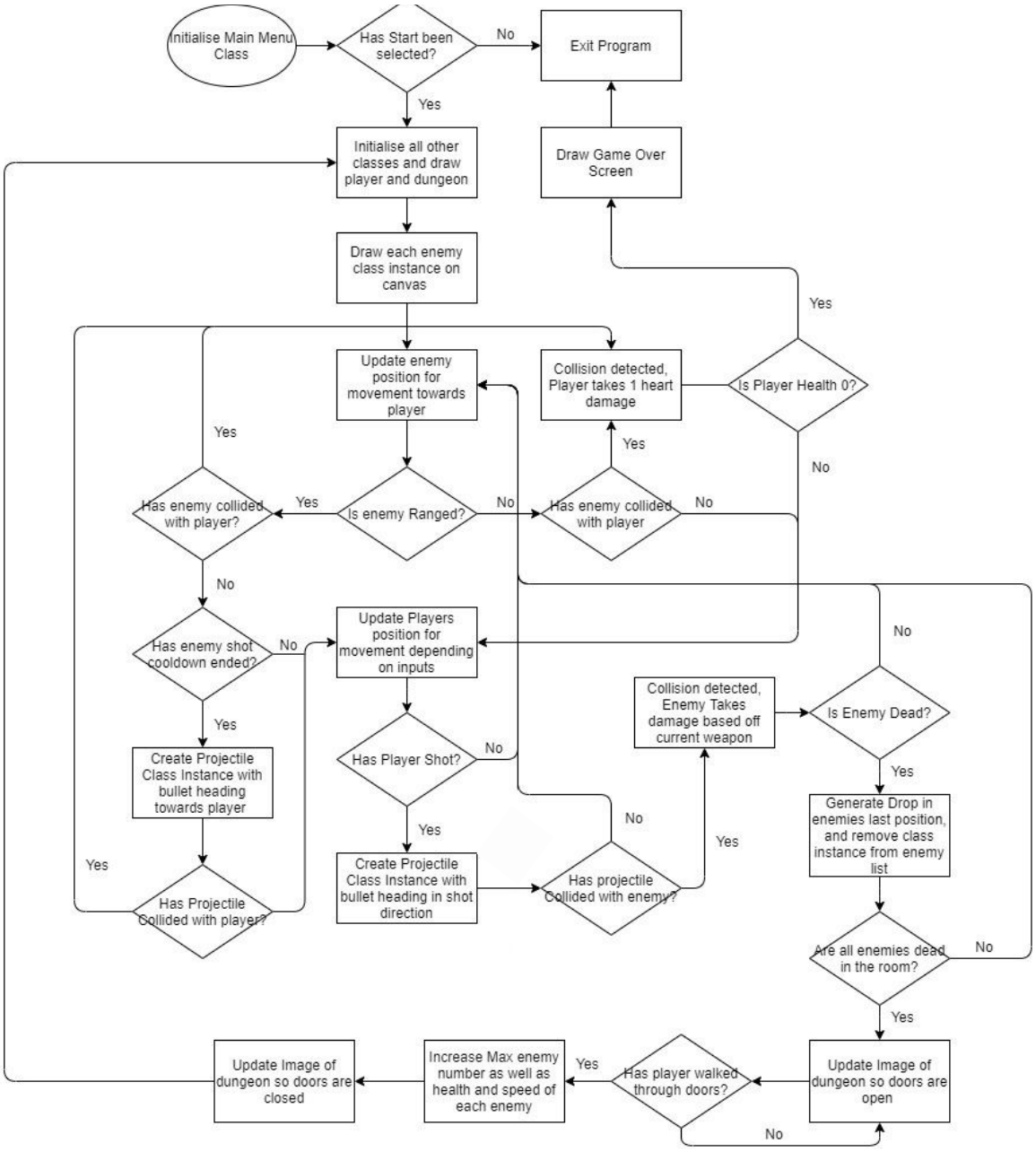
Control Flow

We have 14 different classes in the game as we took a very object oriented approach in the programming. I will describe below very shortly what each of these classes do.

- Interaction Class - The main class of the game that handles the interaction between every other class and contains most game logic
- Game Loop Class - Detect when the player has finished a room or the game for resetting certain methods to their defaults
- Character Class - Contains all the functions used for each character on the screen. Also handles drawing and animation of each character
- Player Class - Inherited from the character class and contains functions only the player uses, like keyboard presses.
- Dungeon Class - Handles drawing for the dungeon.
- Vector Class - Handles coordinate manipulation
- Enemies Class - Inherited from the character class and contains the functions the enemies need to use.
- Ranged_Enemies Class - Inherited from the character class and contains the functions for ranged enemies like the shooting mechanics
- Wall Class - Handles wall collisions.
- Projectile Class - Handles projectile properties and projectile collisions / velocities
- Menu Class - Handles the drawing of the Menu
- Drop Class - Handles the drawing and the functions of each enemy item drop
- Heart Class - Handles the drawing of the players hearts
- Weapons Class - Handles each function of each weapon

Below is a diagram representing the control flow of the main event loop.

Erikas Vieraitis



In short, when the game is launched, the GameLoop Class starts by initialising the Interaction class which contains every other class within it. The Main Menu is drawn from its respective class and then when started, the Interaction class fetches a list of enemies which contain Instances of the enemy class and draws them randomly on the canvas along with the player at the bottom and the dungeon for the background. For each projectile that is shot it creates an instance of it from the Projectile Class, and checks if its a player or enemy projectile. Depending on that variable it then can collide / not collide with its respective entities. When an item is dropped a new instance of a Drop class is created and the player can pick it up. If its a weapon, an instance of a Weapon class is created based on that specific weapon picked up, and its variables decide how it functions when the player has it selected. When all enemies are dead, the GameLoop class re-initialises the Enemy List so new enemies are drawn when the player enters a new room and this loop continues until it detects the player is dead.

Erikas Vieraitis

Code Analysis

In this section I will take apart bits of the most important code in the game and describe how each part of the code works, why I programmed it this way, and how we developed it into the final piece of code it is. I will mainly analyse The Interaction and GameLoop classes as they contain most of the game Logic, as well as a few others.

GameLoop Class

```
class GameLoop:

    def __init__(self):
        self.modifier = 0
        self.ENEMYNUM = random.randint(3,4)
        self.TOP = 65
        self.BOTTOM = 740
        self.heart_pos = 50
        self.projectile_list = []
        self.enemy_list = []
        self.drop_list = []
        self.heart_list = []
        self.walls = [Wall(Vector(self.TOP,self.TOP),Vector(self.TOP,self.BOTTOM)), Wall(Vector(self.TOP,self.BOTTOM),Vector(self.BOTTOM,self.BOTTOM)),
         Wall(Vector(self.BOTTOM,self.TOP),Vector(self.BOTTOM,self.BOTTOM)),Wall(Vector(self.TOP,self.TOP),Vector(self.BOTTOM,self.TOP))]
        self.menu = Menu('https://i.imgur.com/4cnQY7L.png', True, True)
        self.base_gun = Weapons("Pistol", 30, 1, 200, 200, 'https://i.imgur.com/0O1YmHD.png', 0)
        self.player = Player(6, 3, 'https://i.imgur.com/CddMSKn.png', Vector(400, 700), False, self.base_gun, [self.base_gun], 0)
        self.dungeon = Dungeon(1, False)
        self.start_active = False
        self.i = Interaction(self.player, self.dungeon, self.enemy_list, self.walls, self.projectile_list, self.menu, self.drop_list, self.heart_list)


    def draw(self, canvas):
        if self.start_active == False:
            for i in range(self.ENEMYNUM):
                self.enemy_list.append(self.randEnemy())
            for i in range(self.player.hp):
                self.heart_list.append(Heart(Vector(self.heart_pos, 27), False))
                self.heart_pos += 20

            self.start_active = True

        if self.start_active == True:
            if (self.player.dead == True):
                image = simplegui.load_image('https://i.imgur.com/MAQ40Sp.png')
                canvas.draw_image(image, (1000/2, 1000/2), (1000, 1000), (400, 400), (800, 800))
                canvas.draw_text("You made it to Level: " + str(self.dungeon.level), (120, 420), 60, "Orange")
                canvas.draw_text("With " + str(self.player.points) + " points", (250, 480), 60, "Orange")
                canvas.draw_text("Press E to exit the game", (120, 700), 60, "Orange")
                if (self.player.echeck == True):
                    sys.exit()


            else:
                self.i.draw(canvas)
                if (len(self.enemy_list) == 0):
                    self.dungeon.opened = True
                    if (self.player.position.y < 80 and self.player.position.x > 350 and self.player.position.x < 450):
                        self.next_room()
```

Erikas Vieraitis

Above is the GameLoop class which maintains the game's infinite nature. It initialises multiple lists that will hold the projectiles, enemies and drops. It creates an instance of the Wall class, as well as instances of the player, the players starting gun which will be available as soon as they start, the dungeon and the menu. All of those variables in the constructor is put into another class called the interaction class which is what controls all of the interaction for the game.

We have a SimpleGui Draw method which first checks the “start_active” variable. If it’s false it means the player has just started a new game. In this case it will assign enemies to the room randomly using “randEnemy()” and assigns hearts to the player to be drawn using the Heart Class. It then sets “start_active” to true where it won't ever change unless the player starts a new game. If this variable is true, it checks if the player is dead and if he is then it draws the game over screen. If not, it then checks if all enemies in the room are dead. If this is true then it generates the next room using “next_room()” once the players x and y positions are at the door.

```
def next_room(self):
    self.ENEMYNUM += 1
    if (self.ENEMYNUM > 10):
        self.ENEMYNUM = 10
    for i in range(self.ENEMYNUM):
        self.i.enemies.append(self.randEnemy())

    self.dungeon.opened = False
    self.player.position = Vector(400, 720)
    self.i.drop_list = []
    self.i.projectile_list = []
    self.dungeon.level +=1

    if (self.dungeon.level % 5 == 0):
        self.modifier += 1


def randPos(self):
    return Vector(random.randint(70,705),random.randint(85,400))

def randVel(self):
    return Vector(random.randint(1,1),random.randint(1,1))

def randShoot_count(self):
    return random.randint(0,100)
def randEnemy(self):
    selection = [
                Enemies(3 + self.modifier, random.randint(6,8 + self.modifier),
                        'https://i.imgur.com/4m9xLmq.png', self.randPos(), True, self.randVel(), 5),
                Enemies(4 + self.modifier, random.randint(3,4 + self.modifier),
                        'https://i.imgur.com/33dhzaF.png', self.randPos(), True, self.randVel(), 5),
                Enemies(3 + self.modifier, random.randint(4,6 + self.modifier),
                        'https://i.imgur.com/O01BJqL.png', self.randPos(), True, self.randVel(), 5),
                Enemies(5 + self.modifier, random.randint(4,5 + self.modifier),
                        'https://i.imgur.com/x8tNX3p.png', self.randPos(), True, self.randVel(), 10),
                ranged_enemies(3 + self.modifier, random.randint(2,4 + self.modifier),
                               'https://i.imgur.com/xDi6ngE.png', self.randPos(), True,
                               self.randVel(),self.randShoot_count(), 10)
                ]

    return random.choice(selection)
```

This next bit of code contains the remaining 5 methods used within the GameLoop class. The first method is the next_room() method mentioned before. When called it assigns new enemies to the room using the 4 methods below it and sets the drop list and the projectile list to 0 so no projectiles or drops are in the new room. The modifier variable is also increased to boost enemy power

randPos(), randVel() and randShoot_count() all generate random positions, velocities and shooting times respectively using the vector classes

randEnemy() is a method that returns a list of random enemies from the 5 enemy types specified using the Enemies and ranged_enemies classes. This is used every time a new room is entered to have completely random enemies.

Erikas Vieraitis

Interaction Class

```
class Interaction:
    def __init__(self, player, dungeon, enemies, walls, projectile_list, menu, drop_list, heart_list):
        self.player = player
        self.dungeon = dungeon
        self.enemies = enemies
        self.walls = walls
        self.projectile_list = projectile_list
        self.menu = menu
        self.drop_list = drop_list
        self.heart_list = heart_list
        self.shootcount = 1
        self.shoot_active = False
        self.chance = 0
        self.temp_weapon = None
        self.enemy_range = 500
        self.collision_count = 60
```

This interaction class is a class that contains 95% of the game logic. Most of it is within the draw method and it is a very graphical game so I will be going over the most important code in the draw method as well as some of the collision methods.

```
for drops in self.drop_list:
    drops.draw(canvas)
    if (self.player.echeck == True and math.sqrt(((self.player.position.x - drops.position.x)**2)+((self.player.position.y - drops.position.y)**2)) < 40):

        if (drops.name == "Heart"):
            if self.player.hp != 6:
                self.player.hp += 1
                for heart in self.heart_list:
                    if heart.is_empty == True:
                        heart.is_empty = False
                        break
        if (drops.name == "Ammo"):
            self.player.current_weapon.current_ammo = self.player.current_weapon.max_ammo
        if (drops.name == "SuperAmmo"):
            for weapons in self.player.weapon_list:
                weapons.current_ammo = weapons.max_ammo

        if (drops.name == "AK"):
            if len(self.player.weapon_list) == 4:
                self.weapon_swap(Weapons("Ak47", 15, 1, 100, 100, 'https://i.imgur.com/TlsFfxM.png', 0))
            else:
                self.player.current_weapon = Weapons("Ak47", 15, 1, 100, 100, 'https://i.imgur.com/TlsFfxM.png', 0)
                self.player.weapon_list.append(self.player.current_weapon)
            self.weapon_list_modifier()

        if (drops.name == "Desert"):
            if len(self.player.weapon_list) == 4:
                self.weapon_swap(Weapons("Desert Eagle", 60, 3, 35, 35, 'https://i.imgur.com/5O1cZ5l.png', 0))
            else:
                self.player.current_weapon = Weapons("Desert Eagle", 60, 3, 35, 35, 'https://i.imgur.com/5O1cZ5l.png', 0)
                self.player.weapon_list.append(self.player.current_weapon)
            self.weapon_list_modifier()

        if (drops.name == "Mac"):
            if len(self.player.weapon_list) == 4:
                self.weapon_swap(Weapons("Mac11", 10, 0.5, 180, 180, 'https://i.imgur.com/mgorX0n.png', 0))
            else:
                self.player.current_weapon = Weapons("Mac11", 10, 0.5, 160, 160, 'https://i.imgur.com/mgorX0n.png', 0)
                self.player.weapon_list.append(self.player.current_weapon)
            self.weapon_list_modifier()

        if (drops.name == "ShotGun"):
            if len(self.player.weapon_list) == 4:
                self.weapon_swap(Weapons("Shotgun", 40, 1, 80, 80, 'https://i.imgur.com/1H0UeRr.png', 0))
            else:
                self.player.current_weapon = Weapons("Shotgun", 40, 1, 80, 80, 'https://i.imgur.com/1H0UeRr.png', 0)
                self.player.weapon_list.append(self.player.current_weapon)
            self.weapon_list_modifier()
        self.drop_list.remove(drops)
```

Erikas Vieraitis

The code above is part of the draw method within the Interaction class and is used to detect if a player has picked up an item and what item it is. First it goes through every item on the canvas and it checks if the player is within a certain radius and if they have pressed E on the item. If so then it checks through each item that exists in the game. For the heart it adds to the players Hp and checks through each heart displayed in the top left of the screen from left to right. If it detects that one heart slot is empty, it fills it up. If ammo is picked up it just sets the ammo of the gun the player is holding to its max, and the same with super ammo but it does it to all the players weapons. Then there are 4 other weapons you can pick up. For each it first checks if the players weapons slots are full, and if so it calls a method “weapon_swap” which just uses a for loop to go through the players weapons and swap the one they are holding. If it's not full it just adds to the weapons list and the player can use it. After going through all of that it removes the item from the drop list once the player has picked it up so they can’t pick it up again.

```
for enemy in self.enemies:
    for projectile in self.projectile_list:
        if projectile.enemy_projectile == False:
            distance_between = enemy.position.copy().subtract(projectile.pos)
            if distance_between.length() < (enemy.radius + projectile.radius):
                enemy.get_hit(self.player.current_weapon.damage)
                projectile.active = False

        elif projectile.enemy_projectile == True and enemy.can_shoot == True:
            distance_between = self.player.position.copy().subtract(projectile.pos)
            if distance_between.length() < (self.player.radius + projectile.radius):
                if self.player.dead == False and projectile.active == True:
                    projectile.active = False
                    self.heart_depletion()
                    self.player.get_hit(1)
```

This next bit of code is also in the draw method and it is used for handling the interaction of all of the projectiles on screen. It first uses a nested for loop which compares each projectile to each enemy. It then checks if the projectile is an enemy projectile and if it’s not then it uses some vector mathematics to detect if the projectile is within the enemies radius and has collided with an enemy. If so it calls a get_hit method in the enemy class and sets the projectile active to false which removes it from canvas.

If the projectile is a players projectile, then it uses the same vector mathematics to detect if it's in the players radius and if it’s collided with the player, and if it has it calls both heart_depletion method which empties a heart container, and also get_hit which reduces the players health variable.

Erikas Vieraitis

```
def update(self):

    self.collision_count -= 1
    if (self.collision_count < 0):
        self.collision_count = 0

    for enemy in self.enemies:
        enemy.update()
        p_dist_e = enemy.position.copy().subtract(self.player.position)
        unit = p_dist_e.copy().normalize()

        if enemy.can_shoot == True:
            if p_dist_e.length() < self.enemy_range:
                if enemy.enemy_shoot_count % 120 == 0:
                    self.projectile_list.append(enemy.shoot(self.player.position))


            enemy.enemy_shoot_count = enemy.enemy_shoot_count + 1

        enemy.vel = (-Vector(unit.x/3,unit.y/3))
        self.collide(enemy,self.player)
        if self.touching(enemy,self.player) and self.collision_count == 0:
            self.collision_count = 60
            self.player.get_hit(1)
            self.heart_depletion()
        if enemy.dead == True:
            self.player.points += enemy.points
            self.generate_drop(enemy)
            self.enemies.remove(enemy)


    for enemy1 in self.enemies:
        for enemy2 in self.enemies:
            if enemy1 != enemy2 and self.touching(enemy1,enemy2):
                self.collide(enemy1,enemy2)
```

This is the update method in the interaction class. It checks for many things, mainly collisions. This method is called 60 times a second and it first checks through all the enemies in the room. It establishes a distance between each enemy and the player using the variables p_dist_e and unit. It then checks if the enemy can shoot. If so, it checks against the enemies range, and if within it, it creates a projectile launched towards the player.

Next the enemy velocity is determined and it checks if any of the entities are touching. If so, the player takes damage and the collide method takes place where the enemies slightly bounce off. A countdown is also started where the player has 1 second of invincibility. If it detects the enemy is dead it generates a drop and removes it from the canvas.

Finally there is a nested for loop which detects if 2 enemies are touching. If so the collide method is called which makes them slightly bounce off each other.

Erikas Vieraitis

Character Class

```
import simplegui
from user305_wXVUtMiY422VQF4 import Vector

class Character:
    def __init__(self, hp, movement_speed, imageurl, position, is_enemy):
        self.hp = hp
        self.movement_speed = movement_speed
        self.imageurl = imageurl
        self.position = position
        self.is_enemy = is_enemy
        self.index = [0,0]
        self.count = 0
        self.rotation = 0
        self.frame_width = 120
        self.frame_height = 140
        self.radius = 40
        self.dead = False


    def get_hit(self,damage_taken):
        if self.hp > 0:
            self.hp = self.hp - damage_taken
            if self.hp <= 0:
                self.dead = True
        else:
            self.dead = True
    def draw(self, canvas):
        image = simplegui.load_image(self.imageurl)
        canvas.draw_image(image, ((self.frame_width/2)*(self.index[0]+1),
                                  (self.frame_height/2)*(self.index[1]+1)),
                                  (self.frame_width, self.frame_height),
                                   self.position.get_p(), (360/3,420/3), self.rotation)

        if (self.is_enemy == True):
            canvas.draw_line(((self.position.x)-(self.hp * 10), self.position.y-50),
                             ((self.position.x)+(self.hp * 10), self.position.y-50),10,'Red')

    def next_frame(self):
        self.count += 1

        if (self.count % 4 == 0):
            if (self.index[0] <= 4):
                self.index[0] += 2

            if (self.index[0] > 4):
                self.index[0] = 2

                if (self.index[1] <= 4):
                    self.index[1] +=2

                if (self.index[1] > 4):
                    self.index[1] = 0
```

This is the character class, which contains all of the information and algorithms needed for each character entity on screen. This is the parent class of both the player class and the enemy classes too. It handles the animation and the get_hit method of each character.

First is the get_hit method which first checks if the character is not already dead. If not it takes away 1 health point from the character, and declares it dead if health is 0.

Next is the draw method, which draws the character from their sprite sheets. Each sprite sheet is 180x210 pixels and contains 9 different images, so they are split up into their relative frame widths and lengths and drawn like that. The draw method constantly uses the next_frame method which actually animates the character whenever they are moving. With each frame the indexing of the sprite sheet goes from left most to the right most column, for each row until it reaches the end where it is then taken back to the beginning.

Those are some of the main parts of the code. They contain the most technical and most algorithmic code in the game hence why they were chosen. There are still hundreds of more lines of code but these demonstrate the type of skill used to create this game.