# Dungeon Hero

**(Dungeon crawler Top down style game)**

By: Valerie Song, Erikas Vieraitis , Haydn Crossland, Ben Annereau

# Our project - General Overview

- For our project we have decided to pursue a very dominant Object Oriented Approach which includes the use of many classes for most of the individual modules of the game which all interact with one another seamlessly.
- Our game starts off with a Menu which allows the player to choose between 2 options. To start and to exit. This prevents the player from immediately being thrown into the action with no idea what they are doing
- Once the player selects play they are put into a room with multiple enemies. Some which just follow you and try and hit you and some which are ranged and shoot at you from a distance
- Your objective as the player is to eliminate these enemies and keep going through the endless dungeon to see how far you can get. Each room gets harder and harder as the enemies damage and health starts to increase
- The main Source of the players attack is their gun. As the game continues, the player can pick up more powerful weapons and items in the game to help them along their journey and to scale the players power against the enemies power, really testing the players skill
- The player can pick up items from enemies they kill or chests that drop after clearing a room. These items can be replenishing health, ammo, guns or permanent power ups like a speed increase. All these drops are random chance
- The point of this is for the player to keep being pushed and challenged to get further and further into the dungeon, increasing their score as much as possible


# Our aims and objectives

- So when starting the development and planning of this game, our main aim was to make a simple and user friendly game than anyone can play and get immersed in
- We wanted a lot of replayability so we juggled with a few ideas and landed on an endless style of game where the player has to get as far as possible
- From analysing part of the gaming industry we saw that Top down Dungeon crawlers were good for this style so we planned on making a top-down combat style game
- To make a good game you need a variety of enemies and obstacles and one of our aims was to make many different enemies that all act different to give the game some variety
- We wanted the player to actively see their progress so our objective was to make the gameplay progress based where the player is constantly getting stronger the further in the dungeon they go. The aim of this is to create a balanced and fun experience that doesn't get boring the further you go
- To make it progress based we decided the player can pick up a multitude of different weapons and power-ups as they go through the dungeon. Each with different actions and effects for variety
- We aimed to make the game Permadeath where if the player dies they start from the beginning. This style is good as our game is a very randomised chance based game so it adds a lot of replayability each time you play

# Our project - Technical Overview

- Throughout the development of the game we have made use of the vector class. Every movement that is happening on the screen is controlled via the vector class.
- When we were tasked with using this in our game we came to the conclusion that creating a top down view style game can take advantage of things like direct collision and projectiles using the vector class as all axis are available to their full extent.
- This sort of style of game we have lead onto creating gives the player the ability to see all of the action on one screen all at once which creates a more fun and immersive experience

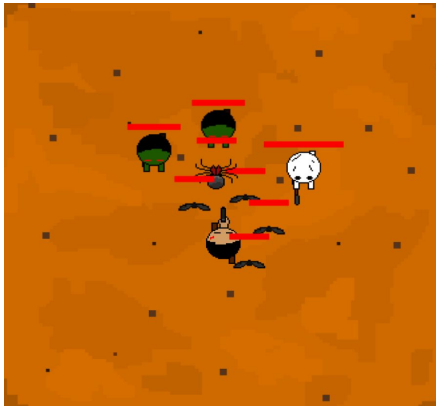

- The physics behind the game use a lot of mathematical logic and quite complicated algorithms which will be explained in the next few slides, but it ensures that enemies and players don't morph into each other and projectiles work properly and can hit each character
- Each object in the game is separated into its own class to make programming as a group much more efficient and easily understandable
- Each character, projectile, drop, and item on screen are separated into their own lists as separate instances of their respective classes.
- We have also taken advantage of super constructors for more efficient and clean code. Each character uses a superclass for the drawing and their animation

# The Classes

- In our game we have 14 Classes that each interact with each other and has their own function to contribute towards the game. We will give a short description of each below

- Interaction Class - This is the main class of the game that handles all of the interaction between every other class and contains most game logic
- Game Loop Class - This classes function is to 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 like a get_hit method and health, movement etc. Also handles drawing of each character on the canvas as well as each animation for each character
- Player Class - Inherited from the character class and contains functions only the player uses, like keyboard presses and weapon switching
- Dungeon Class - Handles drawing for the dungeon and the dungeons open/closed states
- Vector Class - A pre given class that is the baseline for handling coordinate manipulation in the game
- Enemies Class - Inherited from the character class and contains the functions the enemies need to use, like player_hit and alive/dead states
- Ranged_Enemies Class - Inherited from the character class and contains the functions for ranged enemies like the shooting mechanics
- Wall Class - Handles wall collisions for projectiles and characters
- Projectile Class - Handles projectile properties and functions like updating projectile movements, collisions for the projectiles and the drawing of them
- Menu Class - Handles the drawing of the Menu as well as different selection states like if you had Play or Exit Selected it would draw it differently
- Drop Class - Handles the drawing and the functions of each drop item when an enemy is killed
- Heart Class - Handles the drawing of the hearts of the player
- Weapons Class - Handles each function of each weapon. For example the damage, the fire-rate, the ammo and the weapon type as there are 5 different weapons in the game

# Algorithms : Collisions



We did collisions by using three different conditions:

1. If the collision is happening between 2 enemies, we must first check that they are not the same object. This is done using a nested for loop. But if the collision is done between a player an and enemy, it is valid to move to the next condition straight away because the enemy can never be the player and vice versa.
2. If the two sprites sustain to the conditions , we then validate if the length between the two sprites is less than the width of both enemies halved ( essentially the radius)
3. If true, then a method move is called where the two sprites will bounce off each other. This is done by calculating the distance between the two sprites and finding their unit vector.
4. To achieve the bouncing effect we allowed one sprite to take the new velocity of the unit vector, and the other sprite to take the negative unit vector found.

```
def move(self,e1,e2):

    dist_vec = e1.position.copy().subtract(e2.position)
    unit = dist_vec.copy().normalize()
    e1.vel = unit
    e2.vel = -unit
```

```
def update(self):

    for enemy in self.enemies:
        enemy.update()


        p_dist_e = enemy.position.copy().subtract(self.player.position)
        unit = p_dist_e.copy().normalize()
        enemy.vel = (-Vector(unit.x/3,unit.y/3))

        self.collide(enemy,self.player)

        if enemy.dead == True:
            self.generate_drop(enemy)
            self.enemies.remove(enemy)


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

# Algorithms: Projectiles (drawing)

1. A list is used in the main class to handle instances of the projectiles.
2. A separate 'projectiles' class contains the constructor and draw method.
3. The player adds instances of the projectiles to the list of projectiles by using the arrow keys. The velocity given to the projectile is dependant on the key pressed. This allows the projectile to move in the direction of the desired key. The initial position of the projectile depends on the players position. (below)
4. Inside the draw method of main class the list of projectiles is iterated over and the draw method is called. A boolean determines if the projectiles should no longer be drawn, for example if it is off of the canvas or has hit an enemy, these projectiles are then added to a seperate list. (right)
5. This list is iterated over and the projectiles are removed.

```
for projectiles in self.projectile_list:
    projectiles.draw(canvas)
    if (projectiles.active == False):
        projectile_list.remove(projectiles)
```

```
if (self.player.leftcheck == True):
    self.player.rotation = -math.pi/2
    if (self.shoot_active == True):
        self.projectile_list.append(Projectile(Vector(self.player.position.x -55, self.player.position.y), -5, 0, True))
```

# Algorithms: Projectiles (hitting enemies)

1. A list is used in the main class that handles the instances of enemies, with a random number of enemies between 5 and 8 being automatically instantiated. (right)
2. For each enemy, the position of every projectile is checked and the distance between enemies and projectiles are calculated.
3. If this distance between is less than the combined radius of the enemy and projectile, this indicates contact between the two.
4. Once contact has been made, the damage of the weapon is subtracted from the health of the enemy.
5. If the enemy health is below or equal to zero, the enemy is dead. This is indicated with a boolean value. This boolean value is used in the interaction class of the main method to identify which enemies should be removed from the list of enemies.
6. Finally, the projectile is removed from the list of projectiles, regardless of the enemies health. (below)

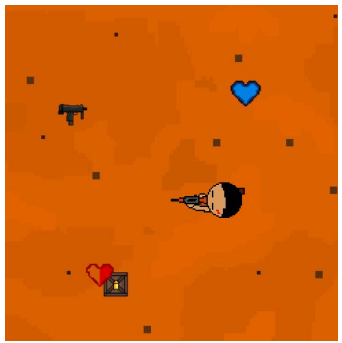
```
ENEMYNUM = random.randint(5,8)
TOP = 65
BOTTOM = 740

projectile_list = []
enemy_list = []
drop_list = []
```

```
for enemy in self.enemies:
    for projectile in self.projectile_list:
        distance_between = enemy.position.copy().subtract(projectile.pos)
        if distance_between.length() < (enemy.radius + projectile.radius):
            if enemy.hp > 0:
                enemy.hp = enemy.hp - self.player.current_weapon.damage
                if enemy.hp <= 0:
                    enemy.dead = True
            else:
                enemy.dead = True
            projectile.active = False
```

# Algorithms: Drops (chance of drop)

1. ‘Drops’ are items that have a random chance of spawning when an enemy is defeated. These items can then be picked up by the player.
2. A list is used to handle the drops
3. When an enemy's health is 0 or below, then the function generate_drop is called
4. The function uses a random integer between 0 and 20 to determine if an item should be dropped, if the integer is 10 or 11, another random is used  to determine which weapon should be dropped.



```
def generate_drop(self, enemy):
    self.chance = random.randint(0,20)

    if (self.chance == 1 or self.chance == 2 or self.chance == 3):
        self.drop_list.append(Drop("Heart", enemy.position, 'https://i.imgur.com/HRAK6TS.png'))


    elif (self.chance == 4 or self.chance == 5):
        self.drop_list.append(Drop("Ammo", enemy.position, 'https://i.imgur.com/p8ucI7W.png'))

    elif (self.chance == 6):
        self.drop_list.append(Drop("SuperAmmo", enemy.position, 'https://i.imgur.com/Dll5Rw4.png'))

    elif (self.chance == 8):
        self.drop_list.append(Drop("BlueHeart", enemy.position, 'https://i.imgur.com/AeOkqCl.png'))

    elif (self.chance == 10 or self.chance == 11):
        randweapon = random.randint(0,3)
        if (randweapon == 0):
            self.drop_list.append(Drop("AK", enemy.position, 'https://i.imgur.com/TlsFfxM.png'))
        elif (randweapon == 1):
            self.drop_list.append(Drop("Mac", enemy.position, 'https://i.imgur.com/mgorX0n.png'))
        elif (randweapon == 2):
            self.drop_list.append(Drop("ShotGun", enemy.position, 'https://i.imgur.com/1H0UeRr.png'))
        elif (randweapon == 3):
            self.drop_list.append(Drop("Desert", enemy.position, 'https://i.imgur.com/5O1cZ5l.png'))
```

# Algorithms: Drops (player pick-up)

1. The list of drops is iterated over and the drops are drawn to the canvas
2. To pick up the item, the user must be within a close proximity and press the ‘e’ key.
3. If it is a power-up, for example ‘ammo’, the players ammo will be completely replenished.
4. If the drop is a weapon, the that will become the players current weapon. Each weapon has its own attributes.

```
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"):
            print("add half a heart to players health")
        if (drops.name == "BlueHeart"):
            print ("add blue heart to players health")
        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"):
            self.player.current_weapon = Weapons("Ak47", 15, 1, 100, 100)
            self.player.weapon_list.append(self.player.current_weapon)
        if (drops.name == "Desert"):
            self.player.current_weapon = Weapons("Desert Eagle", 45, 3, 30, 30)
            self.player.weapon_list.append(self.player.current_weapon)
        if (drops.name == "Mac"):
            self.player.current_weapon = Weapons("Mac11", 10, 0.5, 120, 120)
            self.player.weapon_list.append(self.player.current_weapon)
        if (drops.name == "ShotGun"):
            self.player.current_weapon = Weapons("Shotgun", 40, 1, 40, 40)
            self.player.weapon_list.append(self.player.current_weapon)
        drop_list.remove(drops)
```