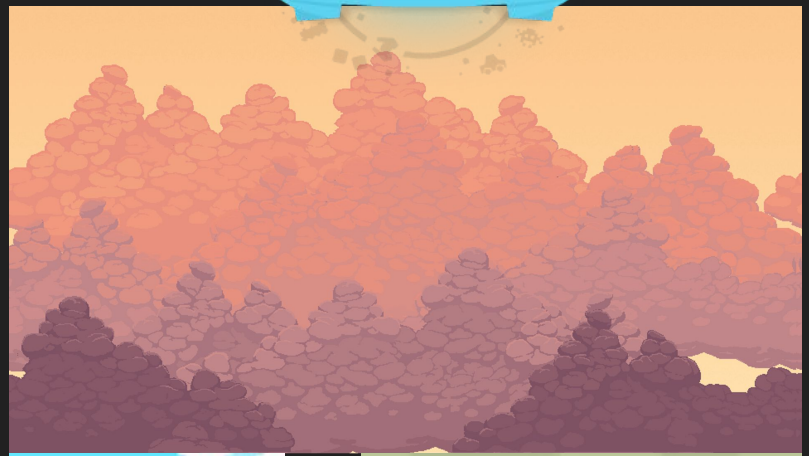
Final Project Presentation

By: Cole Patton and Quentin Hildebrand

What We Learned - Quentin

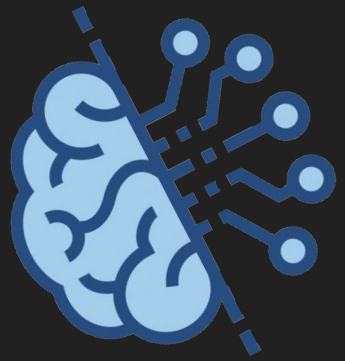
I have learnt while doing this game is that assets that you don't make are pretty hard to find for free and to match the style and genre together. I took me 2-4 classes to change the assets around and even find music and SFX was pretty hard for the gun, the music was hard to match and for it to actually be decent.

The background was probably the easiest to find because it was just a background picture, but the ground was hard because i couldn't find sand or even rocks a bit.



What We Learned - Cole

I learned many things throughout this few month long project. I learned how to write a lot more code in pygame, and how to work with it better. As well, how to work with and implement already existing code into my own code, which really took a lot of understanding of variables and how each piece works with each other!



Besides the code aspect, I also learned a lot of organizational skills for files and pieces of code. I had to keep things sorted so that we knew where everything was, and to be able to quickly access what we needed. Even kept the planning at the start well organized so that we were able to stay on track well.

Successes For Us Both

The successes that we had were simple, but very handy. We were able to find a lot of assets online that worked well for the game we were creating (Kenney assets is a god send). We were also able to successfully find quite a few tutorials and code to help us finalize and create the complete project.

The final success was how great the planning helped. It was nice to have a sort of guideline to follow through on to know where we should be throughout the weeks!

What Went Wrong

The websites wouldn't open some times, the internet was being buggy at school, we got stuck on the code a bit throughout our game, but we would still quickly figure it out. These would all result in some down days where we could not get much done but we powered through.

Besides these simple things, actually not a lot went wrong!

Processes to Get There

To start off, our process was well thought out from the beginning. We used the planning on our scrum board and in our sprints at the start to create a sort of guideline. We followed our board pretty well throughout the games creation, although some of the things we had put down didn't quite make it into the final project either because we were not entirely sure how to implement it, but also because we did not have the time.

From these plannings, we went through using a pygame skeleton and then creating the first bit of code, and then kept making it and following tutorials/stealing code into our game. And that is basically the process of it summed up.

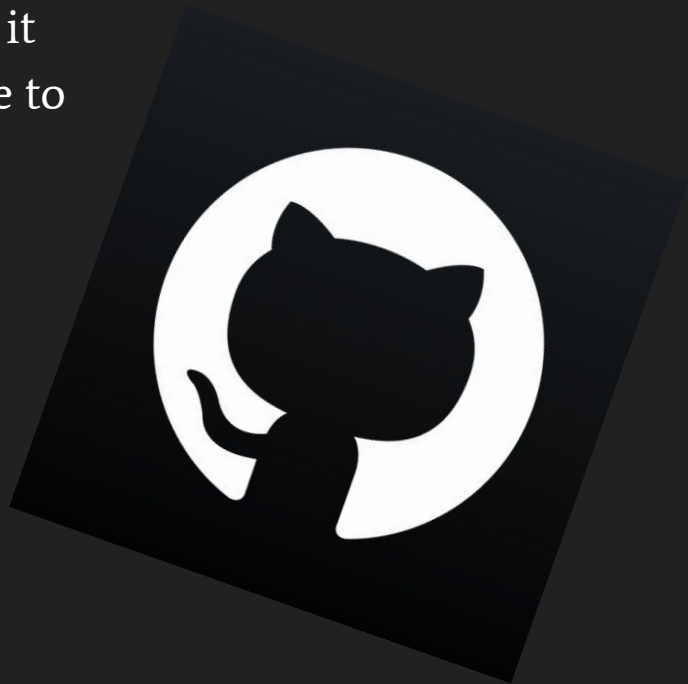
https://padlet.com/colepatton1/dgu2i8rspubue0om
Scrum Board Link^^^^

What We Could Have Improved Upon

There are some definite parts to our project that we could done better on. Parts: like some of the sprints or points on our scrum board that we had originally hoped to include, but never got around to adding. The things like multiple levels, a better replay system, a point system, a highscore etc. All of that had to be scrapped either because we were running outta time or we were not entirely sure how to include it in the code. But for the most part, I think we did pretty well on getting what we really wanted!

What We Could Have Improved Upon

Another last thing we should have done was do better on keeping track of each version of our game using Github. We started doing that but then kinda eventually forgot about it and had just kept progressing so far that it was impossible to keep track of any versions or older version.



What The Next Steps For The Project Would Be

We were thinking that the next steps would be to add what was not NEEDED, but some more things that we would have liked. I know I state this a lot in this presentation but that is definitely one of the things we missed out on. Just some parts that would improve the feel and everything of the game, that wasn't exactly necessary, just nice to have.

We also think that it would have been cool to have our very own assets in the game. Quentin is actually a bit of an artist and is in to that kind of stuff, so if we had more time, it would have been nice to have our very own created original assets in the game.

To sum it up, really the next few steps would simply just be tweaking and improving in small ways that we would have liked!

Complete Game

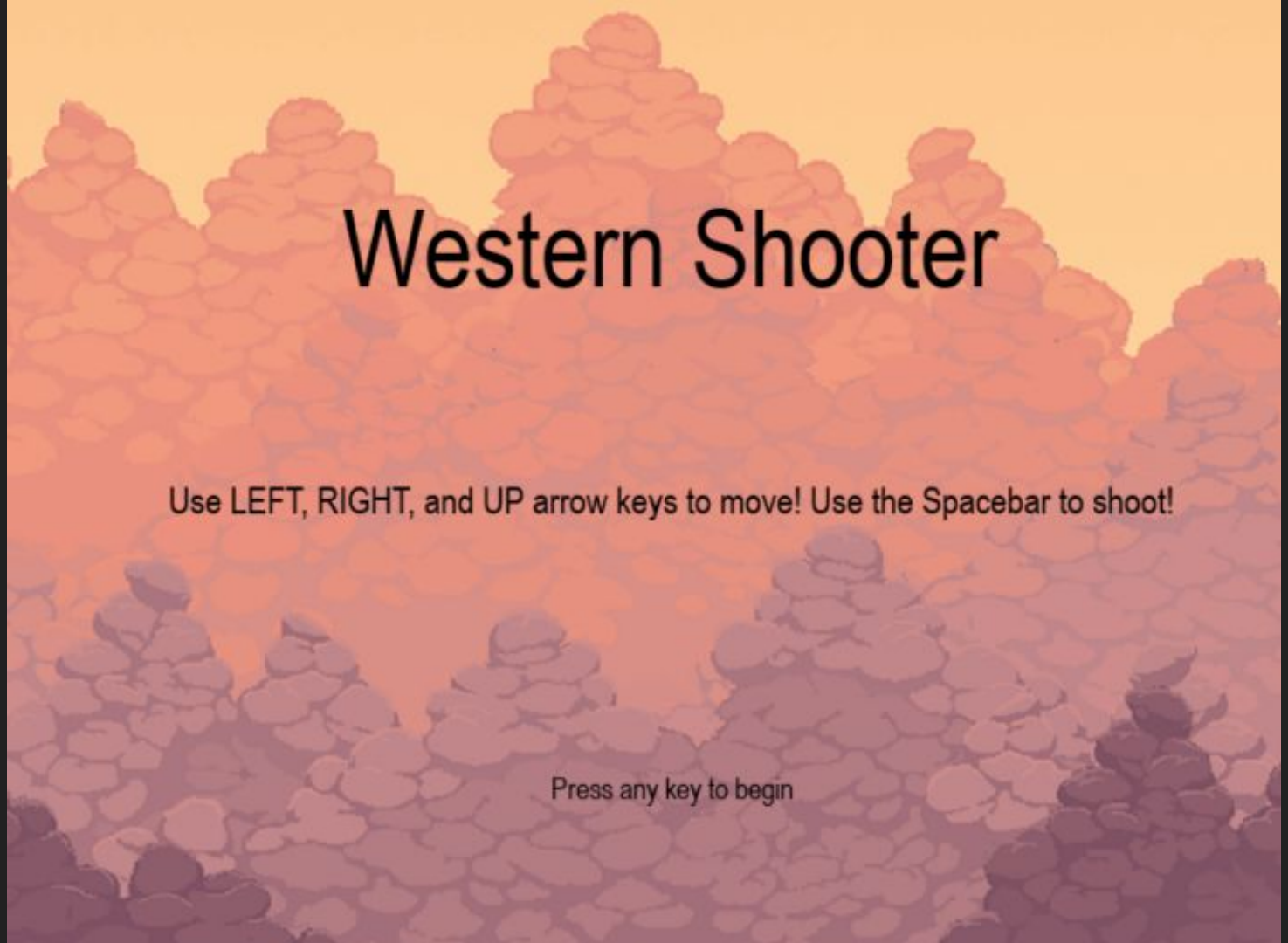
https://github.com/ColePatton/Final-Project-CS30

Here is the link to our Github that has the final game in it, if you are wanting to go take a look and everything! We will not be showcasing off every single line of code since there is a lot of lines, but we will give a good recap of the main pieces of code that are important, as well as our favorite ones that we are even a bit proud of.

Intro Screen

```
#-----------------Functions Defined Here---------------

def intro_screen():
    surface.blit(background, background_rect)
    draw_text(surface, "Western Shooter", 64, width / 2, height / 4)
    draw_text(surface, "Use LEFT, RIGHT, and UP arrow keys to move! Use the Spacebar to shoot!", 22,
              width / 2, height / 2)                                    #Intro text showing the instructions!
    draw_text(surface, "Press any key to begin", 18, width / 2, height * 3 / 4)
    pygame.display.flip()
    waiting = True
    while waiting:
        fps = 60
        clock = pygame.time.Clock()
        clock.tick(fps)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()                   #If statements to see if you quit out, or if any key is pressed,
            if event.type == pygame.KEYUP:      #it will begin the game!
                waiting = False
```


Western Shooter
Use LEFT, RIGHT, and UP arrow keys to move! Use the Spacebar to shoot!
Press any key to begin

Intro Screen

This piece of code was somewhat copied from one of the tutorials (the schmup one) and we reused that piece of code here to add a title screen. It displays the controls, as well as it has the ability to keep the user in the game, but not play it yet, as you can see, you press any key to start the game then. It is pretty simple, but I love my title screens and I like having it in here in our project.

End Screen

```
def exit_window():
    surface.blit(background, background_rect)
    draw_text(surface, "You Win!", 300, width / 2, height / 4)
    pygame.display.flip()
```

End Screen

We figured the next piece of code to showcase would be the end screen after you beat the game, pairing with the intro screen. This too is about as simple as the intro screen, and just displays the two words on the screen right before setting you back onto the main title screen. Pretty handy, it stops the game after the player wins so that player actually knows that they won!

Map + Camera

```
camera_x = 0
camera_walk = 8
```

```
#-------------------------------------------
# This tile map is not only for the camera to move along and follow, but also for
# collision detection of blocks, being able to change the map layout whenever, and
# many more coding things that have to do with the X's.
tile_map = [
    '.....................................................................',
    '.....................................................................',
    '.....................................................................',
    '.....................................................................',
    '.....................................................................',
    '.....................................................xxx.............',
    '.....................................xxxxxx.....xxx.................',
    '.....................................xxxxxxxxxxx.....xxx.............',
    '.....................................xxxxxxxxxxxxxxx.................',
    '.........xxxxxxx..................xxxxxxxxxxxxxxxx.................',
    '.............................xxxxxxxxxxxxxxxxxxxxxx.....xxx.....',
    'xxxxxxxx.........xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    'xxxxxxxxxxx...xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
]
```

Map + Camera

Like what was mostly stated in the code by the comments made, this is the tile map, that is used for MANY things in the code. We used a tutorial for this to help us create a moveable map since we had no idea how that could be created. Basically, we create the map layout in the code in a list like this, and we then use the X's and the .'s to determine what will be blocks (the X's) that the player, enemies, and bullets collide with. And we then use the empty spaces for the .'s for the code to know that there is nothing there, or it is not something that can be collided with.

Map + Camera

This tile map also acts as a sort of treadmill canvas that the screen follows. In the next piece of code, I will explain more and you will see what I am talking about. The camera variable is set to 0, and then as the player walks, the camera variable is either increased (as you move right) or decreased (as you move left). This camera variable then knows the values on when it should be moving the screen to the next set of tiles (or X's) depending on how far the player has moved to either the left or right. It is pretty fascinating, and it worked perfectly for what we wanted to create. As well, we found out that the tile map is very easy to manipulate. We can change the map layout anytime, like expanding upon it, or decreasing it, etc.

Player Controls

```
def handle_movement(self, key):
    global camera_x

    if self.jumping:
        if not(self.would_collide(0, -self.speed, tile_map, True)) and self.jumps_left != 0:
            self.y -= self.speed
        else:
            self.jumping = False
            self.falling = True
            self.jumps_left = self.max_jumps
        self.jumps_left -= 1

    if not(self.would_collide(0, self.speed, tile_map, True)) and not(self.jumping):
        self.falling = True
        self.y += self.speed
    else:
        self.falling = False
```

```
if key[pygame.K_a] and not(self.would_collide(-self.speed, 0, tile_map, True)):
    if self.looking != 'L':
        self.looking = 'L'
        self.image = pygame.transform.flip(self.image, True, False)
    if self.x - self.speed >= 0:
        self.x -= self.speed
    if (self.x % (camera_walk * 50) == 0 or self.x - self.speed < 0) and camera_x - camera_walk >= 0: #Checks to see if there has been enough blocks to m
        camera_x -= camera_walk
        self.x += camera_walk * 50
    elif (self.x % (camera_walk * 50) == 0 or self.x - self.speed < 0):
        self.x += camera_x * 50
        camera_x = 0

if key[pygame.K_d] and not(self.would_collide(self.speed, 0, tile_map, True)):
    if self.looking != 'R':
        self.looking = 'R'
        self.image = pygame.transform.flip(self.image, True, False)
    if self.x + self.speed + self.width <= width:
        self.x += self.speed
    if (self.x % (camera_walk * 50) == 0 or self.x + self.speed + self.width > width) and camera_x + camera_walk + 30 <= len(tile_map[0]):
        camera_x += camera_walk
        self.x -= camera_walk * 50
    elif (self.x % (camera_walk * 50) == 0 or self.x + self.speed + self.width > width):
        self.x -= ((len(tile_map[0]) - 30) - camera_x) * 50
        camera_x = len(tile_map[0]) - 30
```

```
if key[pygame.K_w] and not(self.jumping) and not (self.falling):
    self.jumping = True
```

Player Controls

These player controls are pretty big for walking and jumping, and I will do my best to sum this all up. Any of the lines of code that have the word collision, basically uses the function of determining if you collide, to, well, determine if you collide with a tile. When you walk into something or when the player touches back down to the ground after being in the air.

The next thing these movement lines do is flip the player model depending on which way the use decided to move it. So it flips the model back and forth for you in there, making the visuality of the game that much better.

Colliding Functions

Here are the colliding functions for the player with tiles that I had mentioned in the previous slide.

```
def colliding(rect1, rect2):
    if not (rect1.x + rect1.width <= rect2.x or rect2.x + rect2.width <= rect1.x):
        if not(rect1.y + rect1.height <= rect2.y or rect2.y + rect2.height <= rect1.y):
            return True
    return False

def would_collide(rect1, current_map, camera=False):
    for y in range(len(current_map)):
        for x in range(camera_x if camera else 0, len(current_map[y])):
            if current_map[y][x] == 'X':
                x = (x - camera_x) if camera else x
                map_rect = pygame.Rect(x * 50, y * 50, 50, 50)
                if colliding(rect1, map_rect):
                    return True
    return False
```

Player Controls

The ‘L’ and ‘R’ variables also help determine which way the bullets should be fired depending on the direction that the player is facing. Making the bullets come out of where they should be of course! This is also used in the enemy class, which is a class that inherits the code from the original player class to determine the firing.

And finally, for jumping, it just uses the collision detection to see if you hit the ground or if you hit a wall or a roof, any of those.

Draw Function

```
def draw_window(current_map, player, enemies):
    surface.blit(background, (0, 0))

    for y in range(len(current_map)):
        for x in range(camera_x, len(current_map[y])):
            if current_map[y][x] == 'X':
                if y != 0 and current_map[y - 1][x] != 'X':
                    surface.blit(dirt, ((x - camera_x) * 50, y* 50))
                else:
                    surface.blit(ground, ((x - camera_x) * 50, y *50))
    for bullet in player.bullets:
        rect = pygame.Rect(bullet.x - (camera_x * 50), bullet.y, 10, 3)
        pygame.draw.rect(surface, bullet.color, rect)

    for enemy in enemies:
        for bullet in enemy.bullets:
            rect = pygame.Rect(bullet.x - (camera_x * 50), bullet.y, 10, 3)
            pygame.draw.rect(surface, bullet.color, rect)

        enemy_health_background = pygame.Rect(enemy.x - (camera_x * 50), enemy.y - 20, enemy.width, 10)
        enemy_health_bar = pygame.Rect(enemy.x - (camera_x * 50), enemy.y - 20, enemy.width * (enemy.health / enemy.max_health), 10)

        pygame.draw.rect(surface, RED, enemy_health_background)
        pygame.draw.rect(surface, DARKGREEN, enemy_health_bar)

        surface.blit(enemy.image, (enemy.x - (camera_x * 50), enemy.y))
```

Draw Function

This function essentially draws everything that the main loop needs it to. As well, it uses the loops inside of the function itself to do this, and to keep determining what needs to be drawn. The things like enemies, where the player moves to, the tiles/map layout, and the bullets. It also draws the health bars and other set things that do not need to be in a loop, but is set inside of the draw function so that everything that needs to be drawn is basically in one designated spot.

Enemies + Player

```
player = Player(50, 479, 47, 71, player_image, 'R', 100)

enemies = []
enemies.append(Enemy(500, 379, 47, 71, 3, enemy_image, 'R', 200, 50))
enemies.append(Enemy(1200, 479, 47, 71, 3, enemy_image, 'R', 200, 70))
enemies.append(Enemy(2000, 229, 47, 71, 3, enemy_image, 'R', 300, 70))
enemies.append(Enemy(2900, 479, 47, 75, 3, boss_image, 'R', 200, 720))
```

Enemies + Player

This appending line of code uses an empty list, and an already defined enemy class to create an enemy here and adds however many to a list. It does the same for player, however it only creates one and doesn't append it to any list because that is not needed. Actually one of the main reasons we are putting the enemies into the list is for the thing we will show next.

Determining End

```
if len(enemies) == 0:
    surface.fill(WHITE)
    exit_window()
    pygame.display.update()
    time.sleep(5)
    main()
```

Determining End

This part kind of follows along with the ending screen, but I wanted to share it with the appended enemies so that it is easy to understand. We add the enemies to a list, and at the end of the main function, we create an if statement that checks to see whether or not there are 0 enemies left remaining. Once the if statement becomes true and all of the enemies have been deleted/defeated, then the game exits to the YOU WIN screen, and then relaunches the main function from there, where you have the choice of playing again or quitting out.

Bullet Collision

```
def would_collide(self, x, current_map): #Checks if the bullets collide
    rect = pygame.Rect(self.x + x, self.y, 10, 3)
    return would_collide(rect, current_map)

def touched_enemy(self, enemies):
    for enemy in enemies:
        enemy_rect = pygame.Rect(enemy.x, enemy.y, enemy.width, enemy.height)
        bullet_rect = pygame.Rect(self.x, self.y, 10, 3)
        if colliding(enemy_rect, bullet_rect):
            hurt_sound.play()
            return enemy

    return False
```

Bullet Collision

This function inside of the bullet Class, is for determining if the bullet collides with an enemy or if it instead misses and hits a tile, OR if it just moves on for a certain amount of time, it will be removed and deleted from the game. This is to stop the bullets from just piling up at one end of the screen because that would not be good. This function also then takes the enemies and if the enemy is hit, it will delete some of its health and then updates the enemy health bar. Pretty cool!

The Map + Characters

This last piece of the presentation is just the showcasing here of the map layout and what our enemies and player look like!

You can see here, there is the bullet being shot towards the player on the left side, from the enemy, and there is the health bar, where damage is being done to me the player.
The enemies bullets are red, and the player's bullets are green just to indicate whos is whos.

Here is the first bit of the map!

Here is the second bit of the map! The first part leads into this hill afterwards.

Here is the third bit of the map! Top of the hill, about the center of the map. Should be an enemy here but I had to kill him to survive LOL!

Here is the Last final part of the map where the big bad boss is, he has lots of health. Actually, the enemies progressively have more health the farther you go which is neat! This is the sort of arena area.

Player Character Model

This is the player character model cowboy we found on Kenney Assets. This lil guy is pretty cool and you the player get to control him!

Enemy Character Model

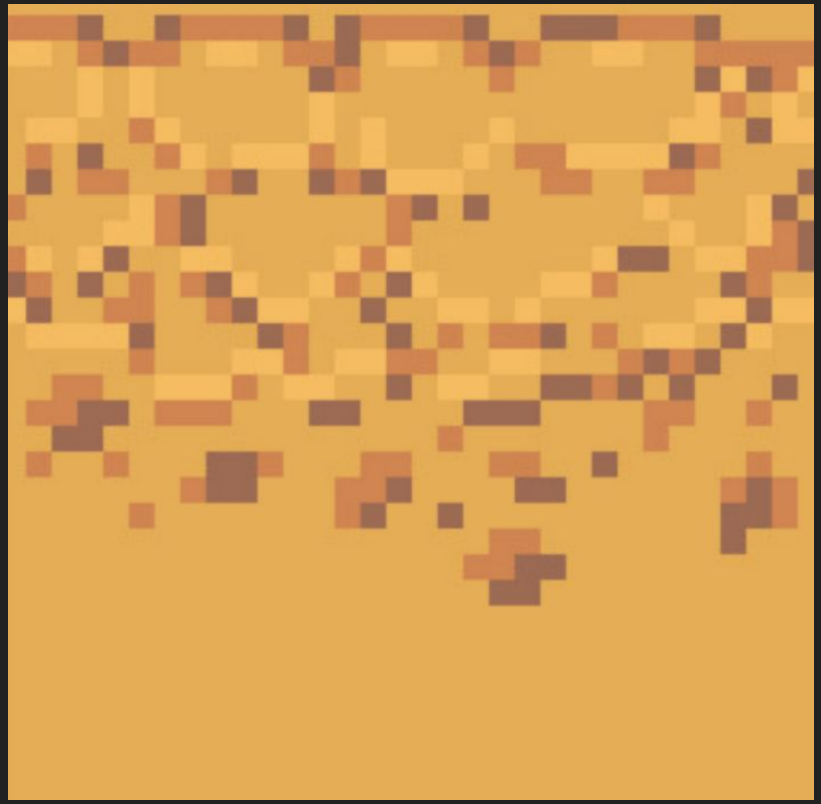
This is the enemy character model, the gangster! He is pretty cool, wielding a lil ole revolver! We found this guy as well in the Kenney Assets pack that we found that had our main character model!

Boss Character Model

THIS is the final boss character model. I (Cole) am a little proud of this one because it looks cool, as well as I sort of made him. What I did was went into Gimp and took the original gangster bad guy and splattered him in red, with a bullet hole in the center of him, and put “blood” or the red pixels all over his hat, and made his whole torso pretty much red too. Pretty cool!

Tile Model

This is the model that we used for the ground and all of the floor in the game. These things are put in place INSIDE of the code as all of the X's!



Background

This is the background we found that we used for the background. Pretty simple looking, but did the job for a hilly sort of background for our western style kind of game!

Sounds Used

We used three different pieces of sounds for this game.

A country sort of western song in the background here:

https://www.youtube.com/watch?v=47l1QLEoWP4

The gunshot sound effect:

https://www.youtube.com/watch?v=f53fti1kwgc

And the hurt/death sound effect stolen from Minecraft:

https://www.youtube.com/watch?v=9iMDchSHfy4

We got these sounds off of YouTube and then uses an mp3 converter to make it a downloadable piece of sound and that also made it usable in our project for calling upon it!

Showcase Ending

That is basically the main pieces of code that we wanted to showcase in this presentation. We would include a video of the game being played from start to finish, but our school laptops have really no way of recording our screen smoothly, and get it here. It is all just a pain, so we figured we would leave that up for you Mr.Whyte! You can try out our little game for yourself! We previously included the link to the Github with the code and all the assets together.

Now of course, we did have to use quite a few different tutorials and pieces of code that other developers had already made, but that is what developers do anyways is steal so we assumed we are good!