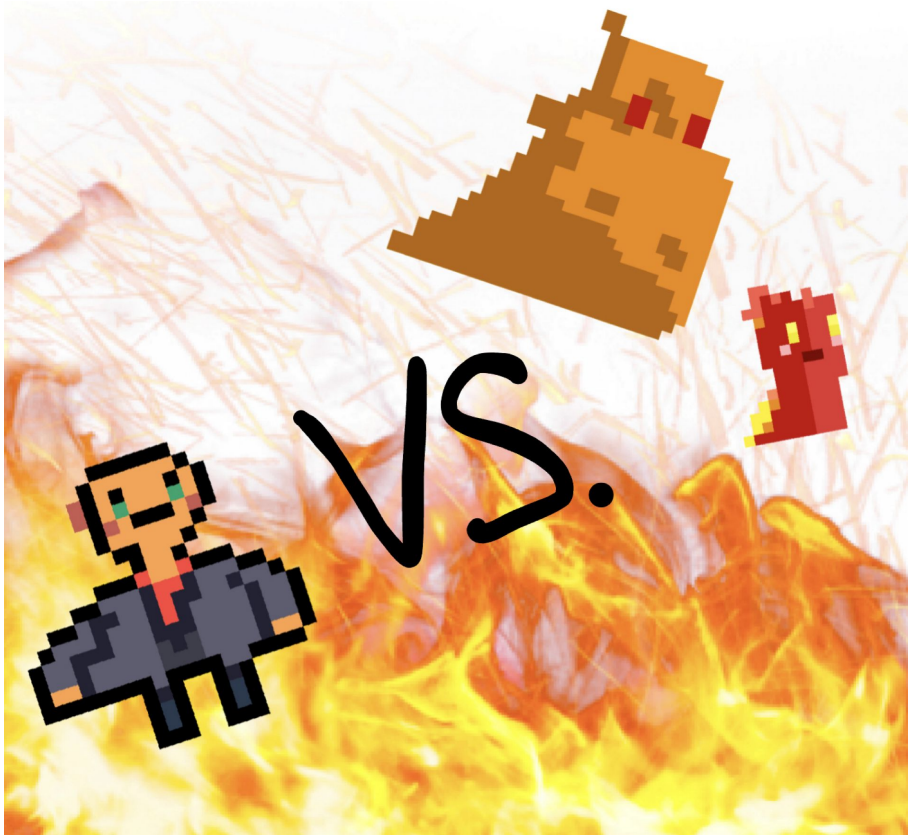
Office Dude vs. Slugs

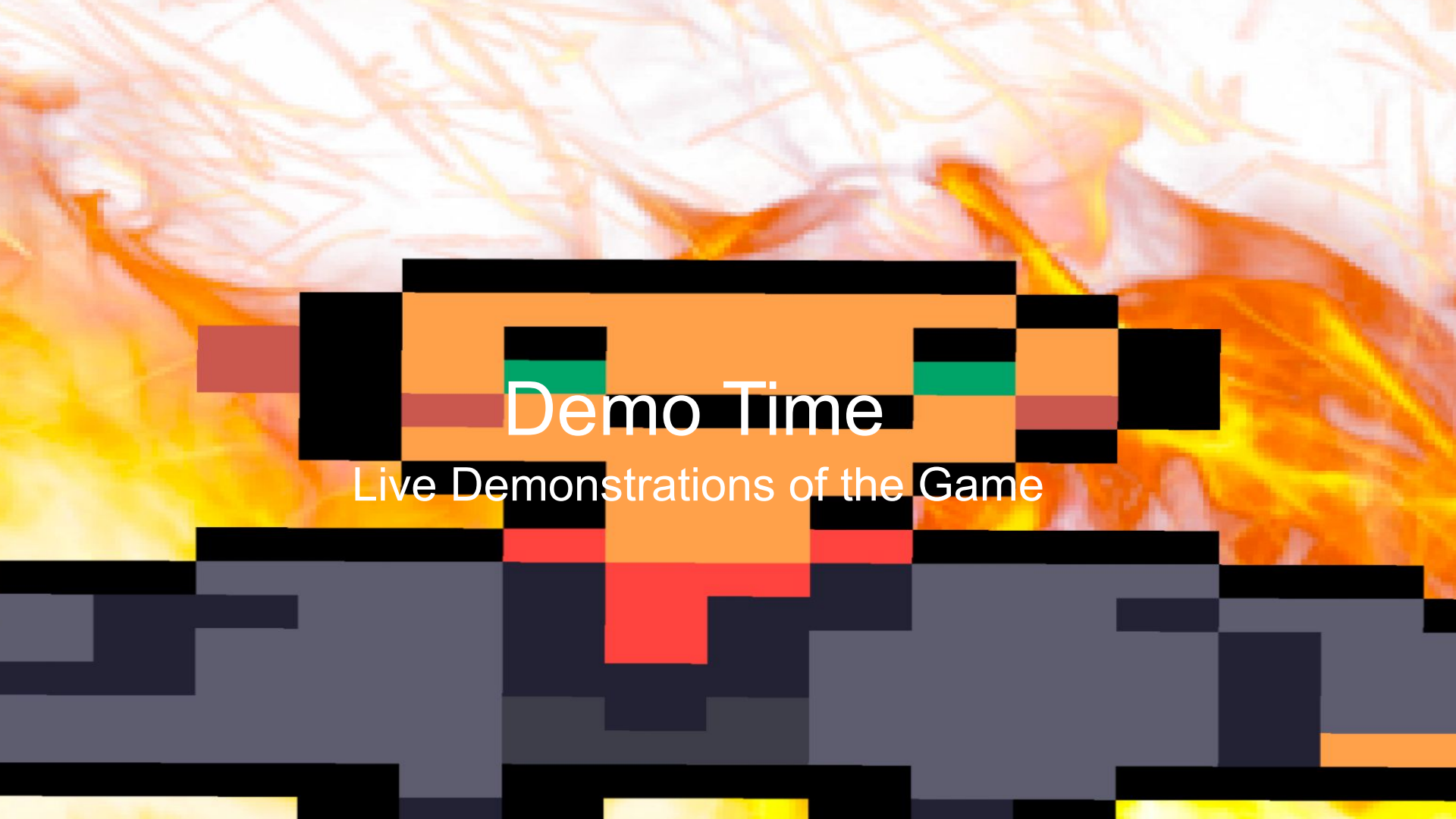
CSC 171 Project

Kyle, Leanna, Kebron and Sunny

Introduction

- Office Dude vs. Slugs is a single player platformer game.
- The player will be fighting against infinite waves of slugs until their health goes to 0.
- Each wave has several slugs for the player to kill. There will be more slugs as the wave count increases.
- Control: WASD. Left click to shoot bullets, long click to shoot larger bullets.




Demo Time
Live Demonstrations of the Game

Canvas

- Extends JPanel
- A massive class that keeps track of all inputs, timers, and positions
- Holds arraylists for all **Colliders**, **Bullets**, **Enemies**, **Blocks**, and **Scores** in the game
- Constructor
  - Adds our **MouseListener** and **KeyListener** inner classes
  - Initializes the arraylist for **Blocks** (builds the walls of the arena)
  - Sets up the physics timer, charge-up timer, and enemy wave timer by instantiating inner classes
  - Creates the player object

Canvas (Inputs)

- Created an inner class called **MouseController** which implements **MouseListener**
  - **mousePressed** - starts charging up a shot
    - Starts our charge-up timer
    - Stores the current time in a long variable
  - **mouseReleased** - fires a bullet
    - Calculates the time elapsed since **mousePressed** (to figure out how powerful to make the bullet)
    - Adds a new **Bullet** to the arraylist, passing in the player's position, the mouse position, and the time elapsed

Canvas (Inputs)

- Created an inner class called **KeyController** which implements **KeyListener**
  - The Canvas class has booleans that store which keys are being pressed currently (W, A, S, D)
    - Set to true in **keyPressed**
    - Set to false in **keyReleased**
    - This way, we aren't calling any physics methods from the **KeyListener** methods themselves
    - Instead, physics methods use the booleans to determine how to move the player
  - Additional keyPressed keys
    - Pressing enter while on the death screen calls our **resetGame** function
    - Pressing escape while on the death screen calls **System.exit(0)** which closes the window

Canvas (Physics - Moving Objects)

- Created a timer and an inner class called Physics which implements ActionListener
- Based on the values of the booleans that hold key inputs (W, A, S, D), the program calls the **Player**'s **setXAccel** and **setYAccel** functions
- Then, the program calls the **updatePosition** function of every moving object in the game (**Player**, **Bullet**, **Enemy**)
    - Uses enhanced for loops on the Bullet and Enemy arraylists

Canvas (Physics - Camera)

- Continuing in the same class Physics which implements ActionListener
- Sets two static variables xCam and yCam to
  - xCam = p.getPosX - (width of screen / 2)
  - yCam = p.getPosY - (height of screen / 2)
  - This centers the camera with the player in the middle of the screen
- How does the camera work?
  - Every object (Player, Enemy, Block) stores its own position coordinates
  - Then, to determine where they should be positioned on the screen relative to the camera, they draw themselves at position - camera position

Canvas (PaintComponent)

- The paintComponent function passes
- It then uses enhanced for loops to draw all **Blocks**, **Walls**, **Enemies**
- It also draws the **Player** and the red laser (which can be toggled on with right click) from the player to the mouse pointer position
- If the player is dead, it draws the UI elements for the game over screen as well

Canvas (Charge-up/Wheel)

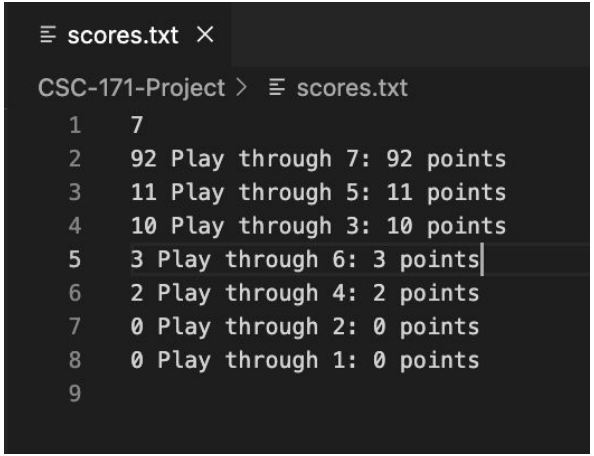
- To make the bullets charge up, we store the current time at the moment the mouse is pressed in a variable
- We also start a timer which uses our inner class **Wheel** which implements **ActionListener**
  - This updates a variable to get the time elapsed since you started holding down the mouse
- The time elapsed is passed in when a new Bullet is made, and the bullet scales its damage and size bigger based on that
- The **Player** draws the wheel with its **drawMe** function with g.fillArc (using the elapsed time to figure out the angle)

Canvas (Wave Spawning)

- To update the waves we have a wave timer which uses an inner class called **Wave** which implements **ActionListener**
- If the **Enemy** arraylist is empty (you killed all **Enemies**) then it will spawn a new wave of enemies
- It is given a number of credits to spend to create enemies
- It randomly spawns new enemies (big enemies cost more credits than small) until it has spent all of its credits
- It then increases the number of credits so the next wave is harder (more enemies can be bought)
- The credits system ensures that every wave is the same difficulty, but you don't get the exact same waves each time

Canvas (Score)

- Whenever the player dies, the screen dims and on the left-hand side of the screen the high score list of the top 5 scores is displayed in descending order
- We have an ArrayList that holds **Score** objects, and sorts the scores in descending order (because the **Score** objects implement comparable and we defined it that way)
- All the data in the score ArrayList is written into a TXT file using **FileWriter**
- When you start up the game again, it reads all the data out of the TXT file back into the ArrayList with **FileReader**
- This way, scores are saved between runs of the game



Main

- Very short class with the main method in it
- Creates a new JFrame and instance of our Canvas class (the main JPanel which holds the elements)

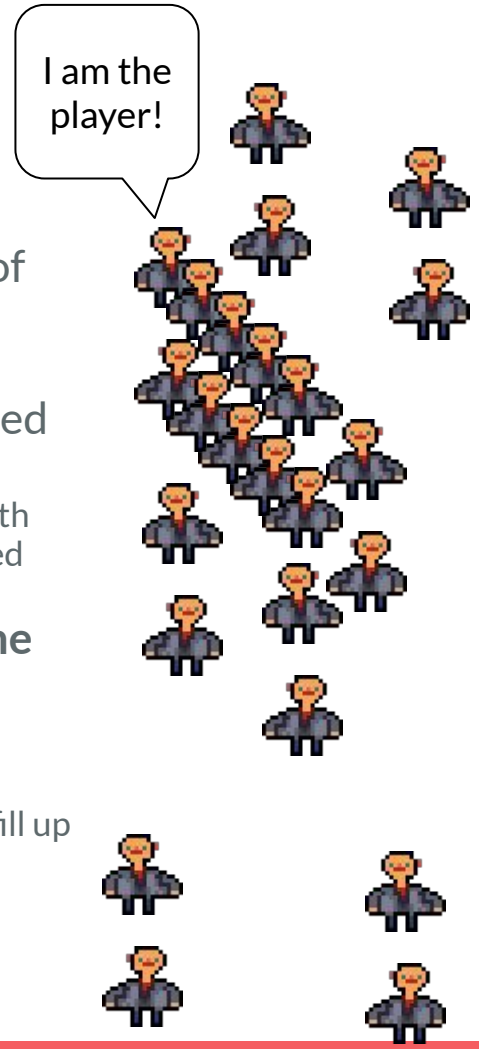
```
public class Main {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Game");
        Canvas canvas = new Canvas();
        frame.add(canvas);
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setVisible(true);
    }
}
```

Collider

- Every object on screen in the program extends the Collider class
- Each object is given a “**tag**”, which is set to the name of their individual class
- Booleans **ignoreBullet**, **hitEnemy**, and **playerHit** are used to determine which object will be damaged
  - ignoreBullet makes sure that the player doesn’t get hit by its own bullet
  - hitEnemy and playerHit set to true if the enemy is hit or the player is hit, and false if vice versa
- If the player is hit, then the player will be damaged; if the enemy is hit, then the enemy will be damaged → possible with the booleans hitEnemy and playerHit

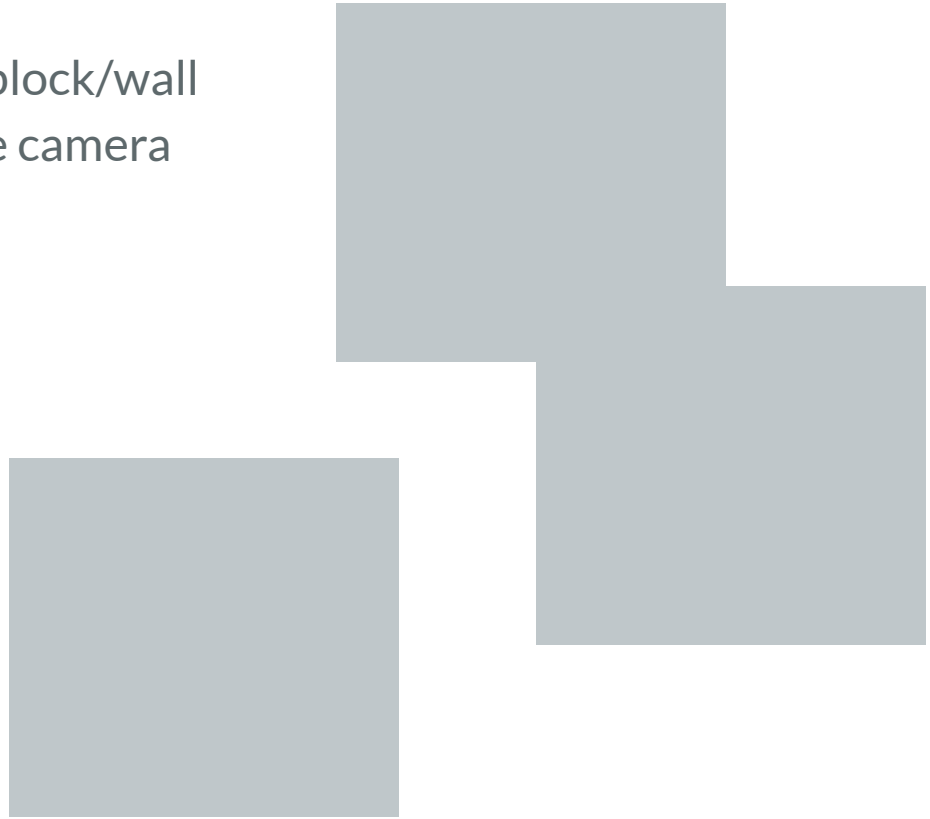
Player

- Takes in **float x** and **float y**; they represent the current location of the player.
- This class manage the **position**, **health**, **collision**, and **damage**
- Invincibility is given to the player with a separate inner class called **Invincibility**
    - Rather than the health always depleting whenever the enemy is colliding with the player, the **timeElasped** variable ensures that the player will get damaged every second
- Draws not only the player itself, but also the **health bar above the player sprite**
    - Done so in the draw() function
- The **charge up wheel** is drawn in the paint component as well
    - Depending on how long the player presses down the mouse, the wheel will fill up in a circular motion accordingly



Block

- Stores x, y, width and height for each block/wall
- Draws itself on the map relative to the camera
- Extends the Collider class

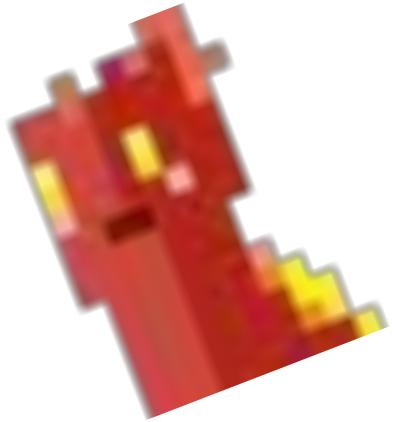


Bullets

- Stores its **x and y position** and the direction vector it should travel in
  - It is spawned at the center of the player
  - The mouse position is passed into the constructor, and it does math to figure out the unit vector in the direction of the mouse
- It also stores **power**, which represents the time elapsed between the moment the mouse is pressed and released
  - The longer the player presses down the mouse, the more powerful the bullet is (increase in size and damage)
- **Extends** the Collider class
  - When a bullet collides with an object, it deletes itself
  - Bullets do not collide with other bullets

Enemy

- This class creates non-stationary objects, enemies, that move towards the player if the player is close to it. Each enemy object is removed from the Array once its health is below 0
- Has boolean method **lineOfSight** that checks if a line from the enemy position to the player position intersects with any walls
- If lineOfSight returns true, the updatePos method will move the enemy towards the player
- **Extends** the Collider class



Score

- 2 parameters: **int score**, **String scoreString**
  - scoreString represents the following phrase:
    - "**Play through " + numberTimesPlayed + ": " + score + " points**"
  - **numberTimesPlayed** is and integer that keeps track of the current player through count
- **Comparable** interface
  - We used this to order the scores by the score variable

Sound

- Has one static method called **playSound()** that takes in a **String s** (the file name)
- Using **try-catch** statements, it tries to search for a file in the directory with the given string
- Then, we use the **AudioInputStream** and **Clip** classes from **javax.sound.sampled** to play the sound

Any Questions?

Thank you for watching!