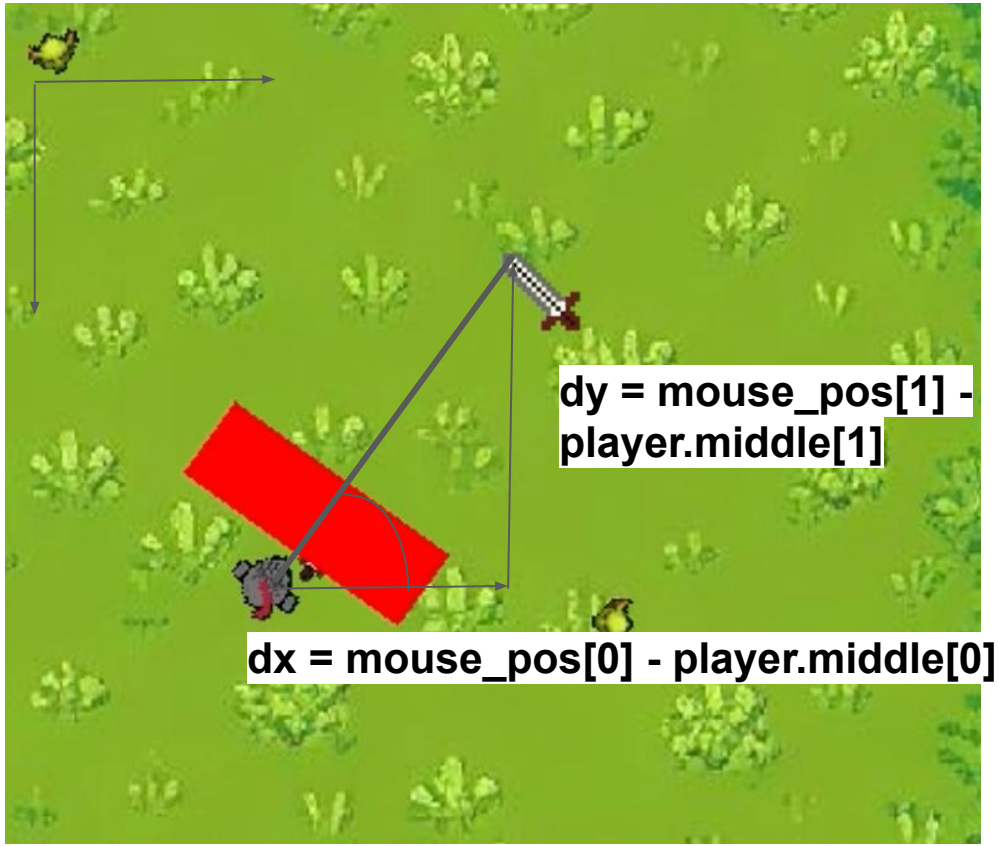
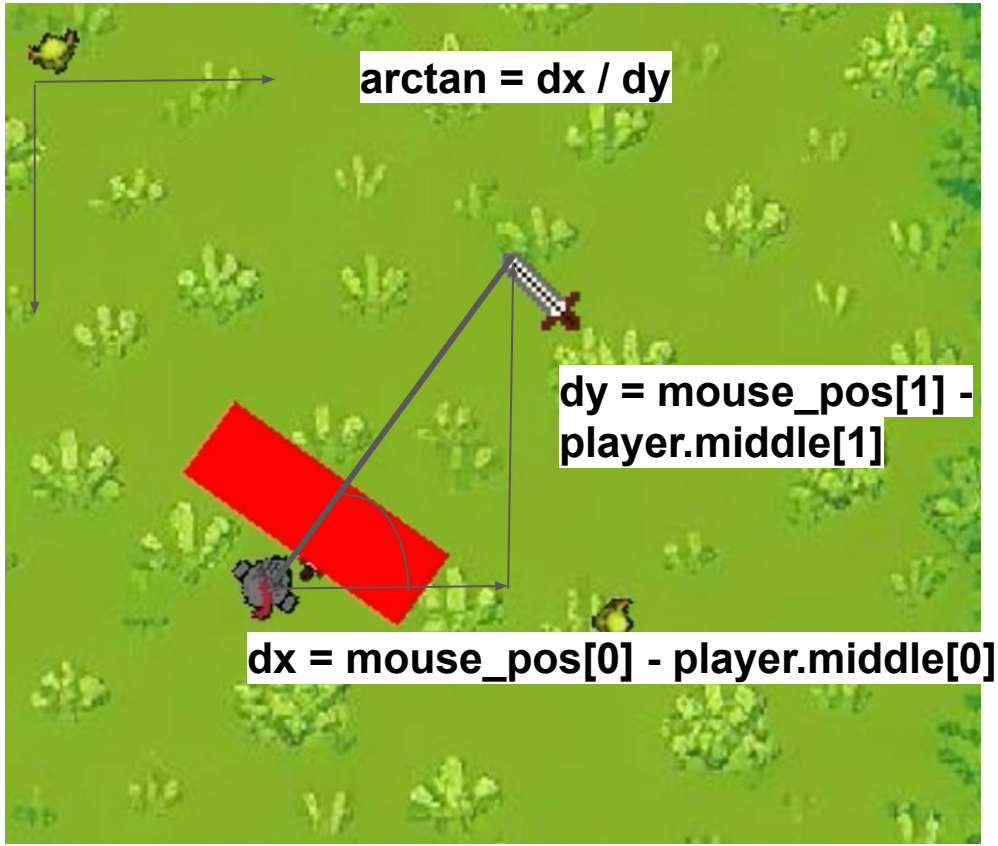
Simon Zwenger
Maximilian Schubert

Attack Range

Attack Range

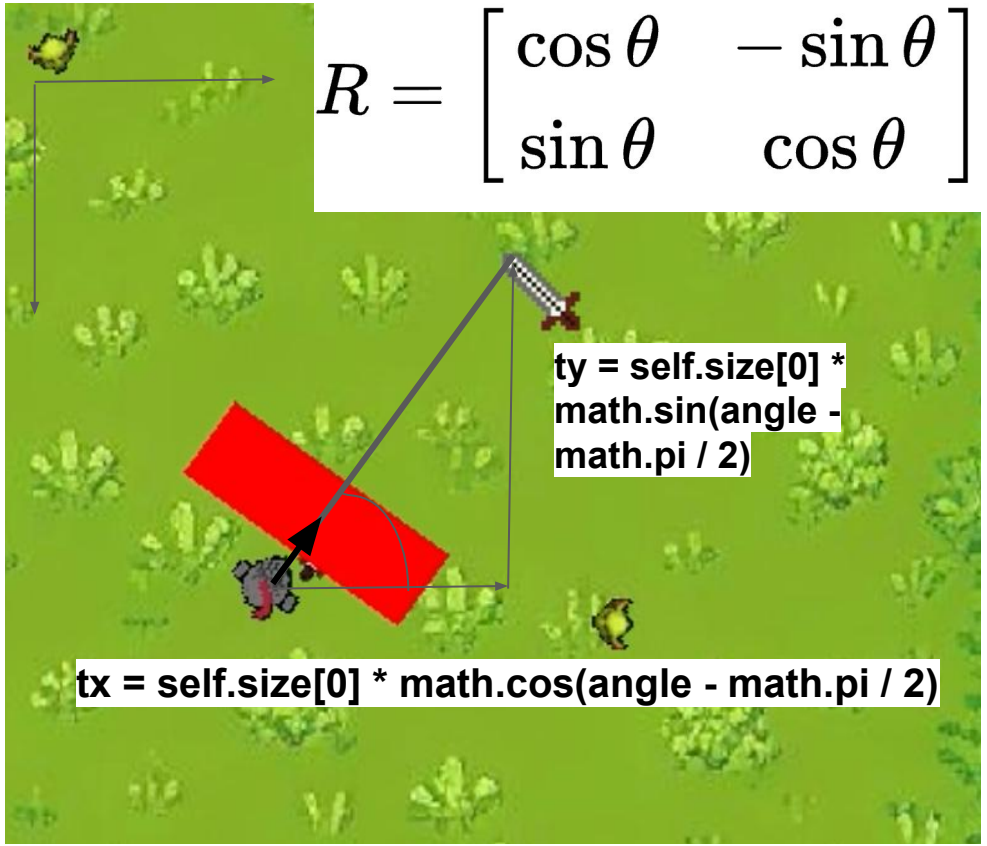


Attack Range



Attack Range

Attack Range



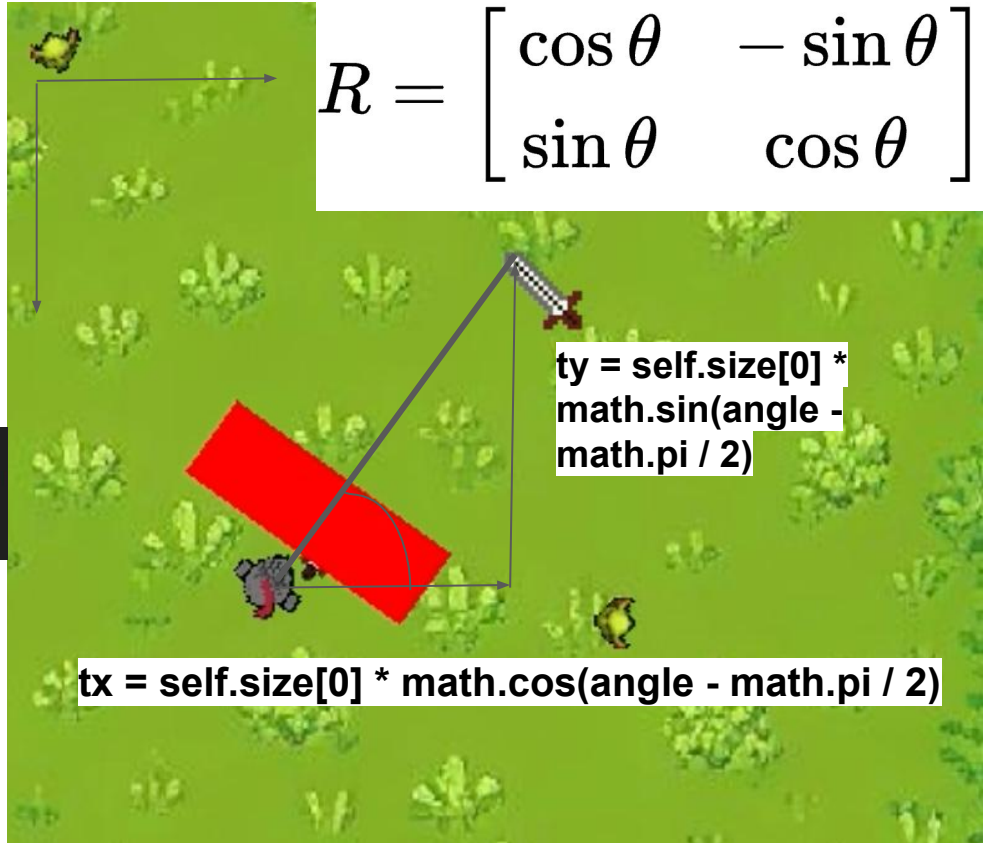
Attack Range

$$\begin{pmatrix} dx \\ dy \end{pmatrix} \times \begin{bmatrix} \cos(\text{angle}) & -\sin(\text{angle}) \\ \sin(\text{angle}) & \cos(\text{angle}) \end{bmatrix} = \begin{pmatrix} dx' \\ dy' \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} dx' \\ dy' \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$

```python
# Calculate the corners of the rectangle
corners = []
for dx, dy in [(-half_width, -half_height), (half_width, -half_height), (half_width, half_height), (-half_width, half_height)]:
    # Rotate and translate the corner relative to the player's position
    x = self.middle[0] + dx * math.cos(angle) - dy * math.sin(angle) + tx
    y = self.middle[1] + dx * math.sin(angle) + dy * math.cos(angle) + ty
    corners.append([x, y])
```

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

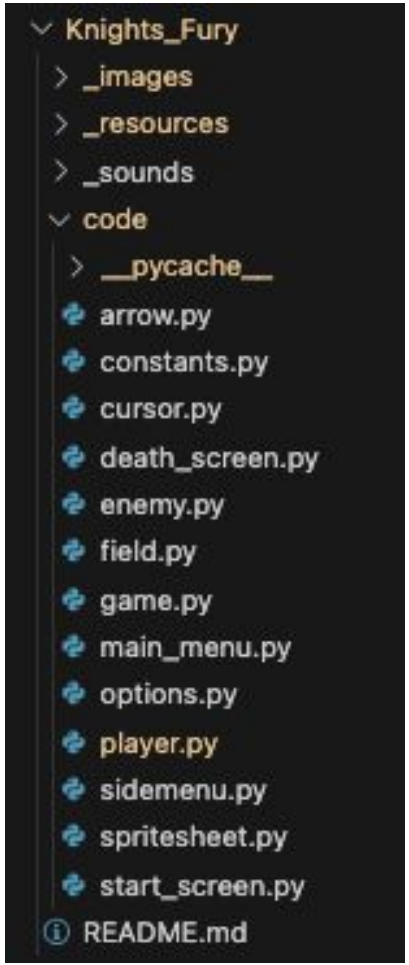


Code for Attack Range

```
def calculate_corners(self, mouse_pos, width, height):
    # Calculate the difference in x and y between the mouse and the player
    dx = mouse_pos[0] - self.middle[0]
    dy = mouse_pos[1] - self.middle[1]
    # Calculate the angle from the player to the mouse, and add 90 degrees
    angle = math.atan2(dy, dx) + math.pi / 2  # Add 90 degrees

    # Calculate half the width and height of the rectangle
    half_width = width / 2
    half_height = height / 2

    # Calculate the translation distance (half the player's size away)
    tx = 2 * self.size[0] / 2 * math.cos(angle - math.pi / 2)
    ty = 2 * self.size[0] / 2 * math.sin(angle - math.pi / 2)

    # Calculate the corners of the rectangle
    corners = []
    for dx, dy in [(-half_width, -half_height), (half_width, -half_height), (half_width, half_height), (-half_width, half_height)]:
        # Rotate and translate the corner relative to the player's position
        x = self.middle[0] + dx * math.cos(angle) - dy * math.sin(angle) + tx
        y = self.middle[1] + dx * math.sin(angle) + dy * math.cos(angle) + ty
        corners.append([x, y])

    # Return the corners of the rectangle
    return corners
```

Code Structure

```python
# ---------- Import Modules and Libaries ----------

import pygame, random, math, time
from constants import * # Import all variables from variabes.py
from main_menu import MainMenu
from start_screen import Start_Screen
from options import Options
from death_screen import Death_Screen
from cursor import Cursor
from player import Player
from arrow import Arrow
from enemy import *
from field import Field
from sidemenu import SideMenu
#from debug import debug

pygame.init() # Initialize Pygame
```



Enemy Class

```python
# Enemy Class
class Enemy:
    def __init__(self, size, speed, weaponsize, attackrange, field:Field):
        #Choose random Spawn-Area
        spawn = random.randint(1,4) # Random Spawn
        if spawn == 1: #top
            self.x = 1*random.randint(0,field.sidelength)
            self.y = -1*random.randint(0,int(round(0.2 * field.sidelength)))
        elif spawn == 2: #bottom
            self.x = 1*random.randint(0,field.sidelength)
            self.y = 1*(random.randint(field.sidelength,int(round(1.2 * field.sidelength))))
        elif spawn == 3:#left
            self.x = -1*random.randint(0,int(round(0.2 * field.sidelength)))
            self.y = 1*random.randint(0,field.sidelength)
        elif spawn == 4: #right
            self.x = 1*(random.randint(field.sidelength,int(round(1.2 * field.sidelength))))
            self.y = 1*random.randint(0,field.sidelength)
```

```python
class SmallGoblin(Enemy):
    def __init__(self, field):
        self.speed = 3
        self.size = 3
        self.delay = 400
        self.health = 1
        self.weaponsize = 45
        self.attackrange = 35
        self.type = "melee"
        super().__init__(self.size, self.speed, self.weaponsize, self.attackrange, field)
```

```python
def get_direction(self, player_pos):…

def enemycollide(self, enemies):…

def inAttackRange(self, player_pos):…

def attack(self, rect: pygame.Rect):…

def update(self, player_pos, rect : pygame.Rect, speed, enemies):
    if self.is_attacking:  # If the enemy is attacking
        if pygame.time.get_ticks() - self.last_attack_time > self.delay:  # If the attack delay has passed
            if self.type == "melee":
                self.attack(rect)  # Per form the attack
                self.is_attacking = False  # Reset the attacking state
    else:  # If the enemy is not attacking
        direction_x, direction_y = self.get_direction(player_pos)  # Get the direction to the player

        self.x += direction_x * speed  # Update the x position of the enemy
        self.y += direction_y * speed  # Update the y position of the enemy

        self.rect.topleft = (round(self.x), round(self.y))  # Update the position of the enemy's rectangle

        self.enemycollide(enemies) # Check for collision with other enemies

        if self.inAttackRange(player_pos):  # If the enemy is in range to attack
            self.is_attacking = True  # Set the attacking state
            self.last_attack_time = pygame.time.get_ticks()  # Set the last attack time to the current time
    self.middle = (self.x + self.size[0] / 2, self.y + self.size[1] / 2) # Update the middle of the enemy
```

Old Graphics