TargetGolf

Kai, Lina, Roy, Thijs

About our Game

- Based on minigolf
- The goal is to get the ball to the target with as few moves as possible
  - The ball moves towards where you click
- There are walls to hinder the path to the target
  - The ball bounces off the walls
- 3 files with around 400 lines in total
- Pygame

Demonstration

Classes.py

- 4 classes in this file:
  - Ball
  - Wall
  - Goal
  - scoreSheet

Ball

```
class Ball:
    def __init__(self, surface, x, y, r = 10, color = (255, 255, 255)):
        # Keep track of the position and velocity in vectors, this makes the physics easier later on
        self.pos = pygame.math.Vector2(x, y)
        self.vel = pygame.math.Vector2(0, 0)
        self.r = r
        self.color = color

        self.screen = surface # pygame wants to know where to draw things, so I added the screen as an argument

        self.friction = 0.005 # this is just a constant to make the ball slow down over time, I put it here so it's easy to tweak if needed
```

- Shape
- Physics
  - Movement
  - Velocity
  - Acceleration
  - Collision/bounce
    - Interaction with the walls
- draw

Wall

```
class Wall:
    def __init__(self, surface, x, y, orientation, length, thick=10, color=(255, 100, 0)):
        if orientation == "vertical":
            width = thick
            height = length
        elif orientation == "horizontal":
            width = length
            height = thick
        self.rect = Rect(x, y, width, height)
        self.orientation = orientation
        self.color = color
        self.screen = surface

    def draw(self):
        pygame.draw.rect(self.screen, self.color, self.rect)
```

Goal

- Positioning
- Shape
- Scoring

```
class Goal:
    def __init__(self, surface, x, y, r = 10, color = (0,0,0)):
        self.pos = pygame.math.Vector2(x,y)
        self.r = r
        self.color = color
        self.screen = surface

    def draw(self):
        x = int(self.pos.x)
        y = int(self.pos.y)
        pygame.draw.circle(self.screen, self.color, (x,y), self.r)

    def hit(self,ball):
        # Thijs: your code correct, it was just in the wrong spot with the event handling, so I moved it over here and added returns.
        distance = math.hypot(ball.pos.x-self.pos.x,ball.pos.y-self.pos.y)#Roy: I added this line so it updates the distance everytime we move the ball
        if distance <= (ball.r+self.r): #Roy: This is the if-statement that checks for collision, e.g. if the ball hits the target
            self.color =  (0,0,255) #Roy: for now, I just told the code to change the color of the goal, we have to see what we want the program to do once the goal
            return True
        else:
            return False
```

Scoresheet

- Scoring per level
- Makes a table with levels and scores for each

```
class scoreSheet():
    def __init__(self, screen):
        screenWidth, screenHeight = screen.get_size()
        self.strokes = []
        self.screen = screen
        self.screenWidth = screenWidth
        self.screenHeight = screenHeight
        self.width = 250
        self.height = 510
        self.font = pygame.font.SysFont('arial', 22)
        self.bigFont = pygame.font.SysFont('arial', 30)
```

Levels.py

- Levels are made with one generic level parent class
  - Uses ball, goal and wall classes
- Each level is a child class with different layouts of the walls

```
class Level:
    #Generic super-class used to define a level. For each level with level"
    #specific info we should create a separate child class"

    def __init__(self, screen):

        # Lists of sprites used in all levels.
        self.background = None   #In parent class none; we can change this per level
        self.screen = screen
        self.finished = False
#         self.goal = goal

        framerate = 60 # I set this framerate to use later

        self.b = None#Ball(screen, screenWidth/2, 500)
        self.g = None#Goal(screen, 400, 100)
        self.walls = []

    def update(self):
        self.b.bounce(self.walls)
        self.b.move()
        if self.g.hit(self.b):
            self.finished = True

    def click(self, mousePos):
        acc = (mousePos - self.b.pos).normalize()
        acc *= 4
        self.b.accelerate(acc)

    def draw(self):
        self.g.draw()
        for wall in self.walls:
            wall.draw()
        self.b.draw()
```

Child classes - individual levels

- Background colour
- Positioning of ball & goal
- Positioning of walls

```
class Level_01(Level):
    #definition for level 1

    def __init__(self, screen):
        Level.__init__(self, screen) # Call the parent constructor
        screenWidth, screenHeight = screen.get_size()
        self.background = (98, 166, 92)
        self.b = Ball(screen, screenWidth/2, 500)
        self.g = Goal(screen, 400, 100)

        self.walls = [Wall(screen, 0, 0, "horizontal", screenWidth),
                 Wall(screen, 0, 0, "vertical", screenHeight),
                 Wall(screen, screenWidth-10, 0, "vertical", screenHeight),
                 Wall(screen, 0, screenHeight-10, "horizontal", screenWidth),
                 Wall(screen, 0, screenHeight/3, "horizontal", screenWidth*2/3),
                 Wall(screen, screenWidth/3, screenHeight*2/3, "horizontal", screenWidth*2/3)]
```

Main.py

- Main game loop
  - Plays the levels one by one
  - Ball movement by clicking
  - Keeps track of score

```
        # This for loop checks the pygame events: things like user inputs get turned into those events
        for event in pygame.event.get():
            # This event handles close the window
            if event.type == QUIT:
                loop = False
                # This event handles when the user quits
            if event.type == MOUSEBUTTONDOWN:
                mousePos = pygame.math.Vector2(event.pos)
                current_level.click(mousePos)
                strokes += 1

        screen.fill(current_level.background) # set a background on top of everything that was drawn last frame

        # Update the physics
        current_level.update()

        # Draw all the objects
        current_level.draw()

        if current_level.finished:
             sheet.drawSheet(strokes)
             strokes = 0
             current_level_no += 1
             current_level = level_list[current_level_no]
             wait = True
             while wait:
                 for event in pygame.event.get():
                     if event.type == MOUSEBUTTONDOWN:
                         wait = False


        # Update the screen and wait for the next frame (without the delay the game runs way too fast)
        pygame.display.update()
        pygame.time.delay(int(1000/framerate)) # The delay function takes a time in milliseconds, the framerate is

    # If we leave the game loop; quit
    pygame.quit()
```