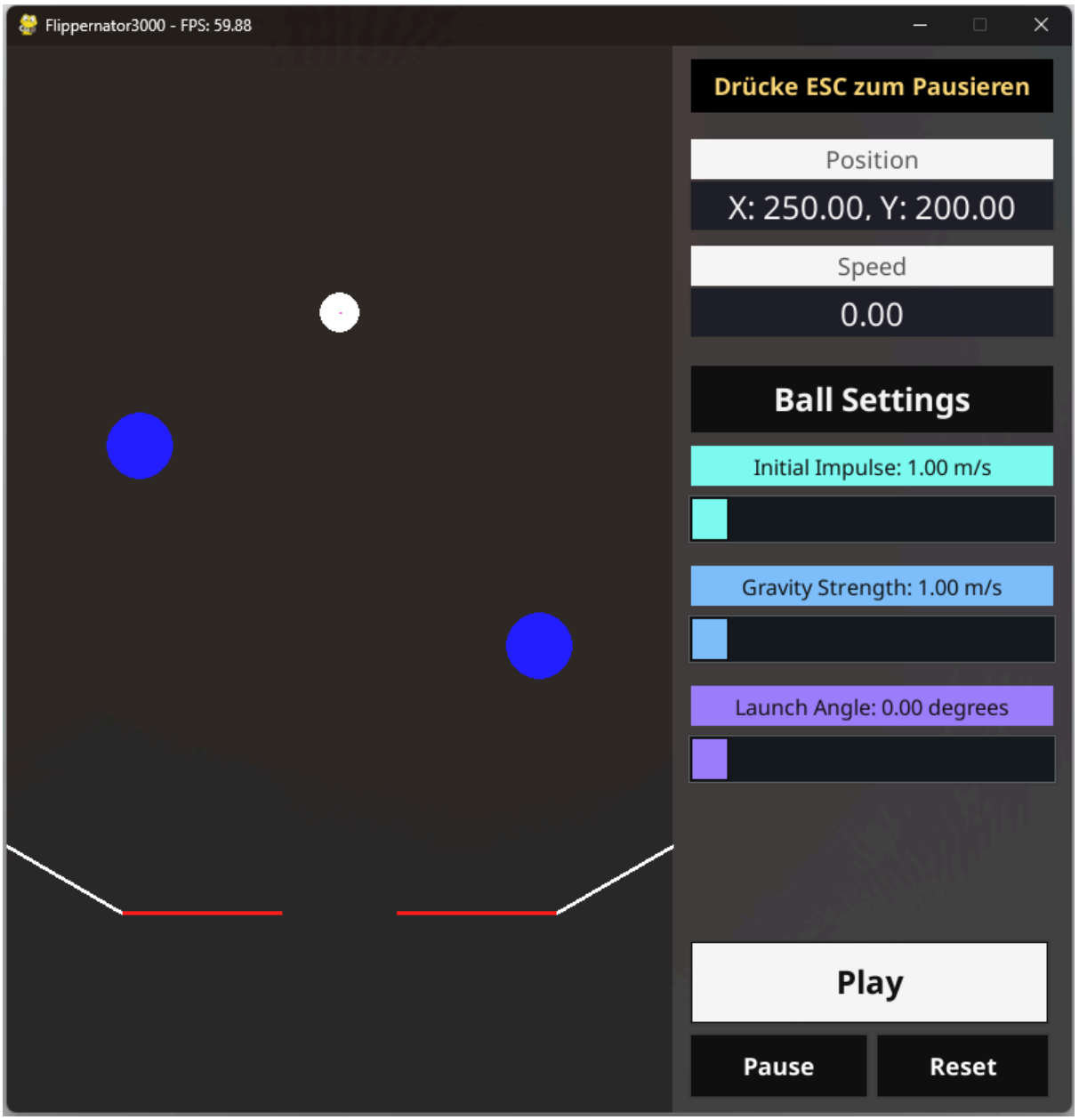
VC II // Flipper Dokumentation

Team: Flippernator3000, Vanessa Frey, Julian Gutknecht

Abgabe 2



Code Snippets

Rollen auf der schiefen Ebene

Die “apply_flipper_physics”-Funktion simuliert das Rollen der Kugel auf einer schiefen Ebene (in diesem Fall die Flipper). Sie berechnet die Komponente der Schwerkraft, die entlang des Flipperwinkels wirkt, und aktualisiert entsprechend die Geschwindigkeit der Kugel.
Die weiteren Funktionen ermöglichen es der Kugel, auf den schiefen Ebenen (Rampen) zu rollen. Die “check_ramp_collision”-Funktion überprüft Kollisionen zwischen der Kugel und den Rampen, und die reflect_ball-Funktion reflektiert die Kugel basierend auf dem Normalenvektor der Rampe.

```
def apply_flipper_physics(ball_pos, ball_vel, flipper_start, flipper_end):
    flipper_angle = math.atan2(flipper_end[1] - flipper_start[1], flipper_end[0]
- flipper_start[0])
    gravity_parallel = GRAVITY * math.sin(flipper_angle)

    # Aktualisiert die Geschwindigkeit der Kugel basierend auf der Schwerkraft
entlang des Flipperwinkels
    ball_vel[0] += gravity_parallel * dt * math.cos(flipper_angle)
    ball_vel[1] += gravity_parallel * dt * math.sin(flipper_angle)

    # Berechnet die projizierte Position der Kugel nach der Geschwindigkeit
    projected_pos = [ball_pos[0] + ball_vel[0] * dt, ball_pos[1] + ball_vel[1] *
dt]

    # Berechnet den Abstand der projizierten Position zum Flipper
    dist_to_flipper = point_line_distance(projected_pos, flipper_start,
flipper_end)

    # Wenn die Kugel den Flipper verlässt, korrigiert die Position, damit sie
darauf bleibt
    if dist_to_flipper > BALL_RADIUS:
        normal = get_line_normal(flipper_start, flipper_end)
        ball_pos[0] -= normal[0] * (dist_to_flipper - BALL_RADIUS)
        ball_pos[1] -= normal[1] * (dist_to_flipper - BALL_RADIUS)

...

def check_ramp_collision():
    global ball_pos, ball_vel

    # Durchläuft alle Rampen und überprüft, ob die Kugel eine Kollision hat
    for ramp_start, ramp_end in [(ramp_left_start, ramp_left_end),
(ramp_right_start, ramp_right_end)]:
        if point_line_distance(ball_pos, ramp_start, ramp_end) <= BALL_RADIUS:
```

```
            reflect_ball(ramp_start, ramp_end)
            break

def reflect_ball(start, end):
    global ball_angular_vel

    # Berechnet den Normalenvektor der Linie
    normal = get_line_normal(start, end)
    midpoint = ((start[0] + end[0]) / 2, (start[1] + end[1]) / 2)
    ball_to_midpoint = (midpoint[0] - ball_pos[0], midpoint[1] - ball_pos[1])

    if (ball_to_midpoint[0] * normal[0] + ball_to_midpoint[1] * normal[1]) > 0:
        normal = (-normal[0], -normal[1])

    ball_vel[0], ball_vel[1] = reflect(ball_vel, normal)

    # Wendet Drehmoment basierend auf der Kollision an
    collision_vector = [ball_pos[0] - midpoint[0], ball_pos[1] - midpoint[1]]
    torque = (collision_vector[0] * ball_vel[1] - collision_vector[1] *
ball_vel[0]) / (BALL_RADIUS ** 2)
    ball_angular_vel += torque

    # Stellt sicher, dass die Kugel nicht zu weit in das Objekt eindringt
    while point_line_distance(ball_pos, start, end) <= BALL_RADIUS:
        ball_pos[0] += normal[0] * 0.1
        ball_pos[1] += normal[1] * 0.1
```

Abhängigkeit von Reibung und Bewegung

Diese Funktion bewegt die Kugel unter Berücksichtigung von Schwerkraft und Dämpfung (Reibung). Wenn die Geschwindigkeit der Kugel unter einen bestimmten Schwellenwert fällt, wird die Kugel gestoppt.

```
def move_ball():
    global GRAVITY, INITIAL_BALL_IMPULSE, BUMPER_BOUNCE, ball_angle,
ball_angular_vel, ball_vel, ball_pos

    if not GAME_STARTED:
        return

    # Default
    if ball_vel == [0, 0]:
        angle_rad = math.radians(BALL_ANGLE + 90)
        ball_vel[0] = INITIAL_BALL_IMPULSE * math.cos(angle_rad)
        ball_vel[1] = INITIAL_BALL_IMPULSE * math.sin(angle_rad)

    # Fügt die Schwerkraft hinzu
    ball_vel[1] += GRAVITY * dt

    # Aktualisiert die Position der Kugel
    ball_pos[0] += ball_vel[0] * dt
    ball_pos[1] += ball_vel[1] * dt

    # Aktualisiert die Winkelposition
    ball_angle += ball_angular_vel * dt

    # Wendet Dämpfung an
    ball_vel[0] *= DAMPING_FACTOR
    ball_vel[1] *= DAMPING_FACTOR
    ball_angular_vel *= DAMPING_FACTOR

    # Überprüft, ob die Geschwindigkeit unterhalb des Schwellenwerts liegt, und
stoppt die Kugel, wenn ja
    if math.hypot(ball_vel[0], ball_vel[1]) < VELOCITY_THRESHOLD:
        ball_vel = [0, 0]
        ball_angular_vel = 0

    # Begrenzt die Position der Kugel auf die Spielfeldgrenzen
    ball_pos[0] = max(min(ball_pos[0], GAME_WIDTH - BALL_RADIUS), BALL_RADIUS)
    ball_pos[1] = max(min(ball_pos[1], HEIGHT - BALL_RADIUS), BALL_RADIUS)
```

Kollisionserkennung (Flipper & Spielfeld)

Diese Funktion überprüft Kollisionen der Kugel mit den Flippern und den Spielfeldgrenzen. Bei einer Kollision wird die Geschwindigkeit der Kugel entsprechend reflektiert.

```
def check_collision():
    global ball_pos, ball_vel

    # Überprüft Kollisionen mit den Flippern
    for flipper_pos, angle, is_right in [(left_flipper_pos, left_flipper_angle,
False), (right_flipper_pos, right_flipper_angle, True)]:
        start_x, start_y = flipper_pos
        end_x = start_x + FLIPPER_LENGTH * math.cos(math.radians(angle)) * (-1
if is_right else 1)
        end_y = start_y - FLIPPER_LENGTH * math.sin(math.radians(angle))

        flipper_start = (start_x, start_y)
        flipper_end = (end_x, end_y)

        if point_line_distance(ball_pos, flipper_start, flipper_end) <=
BALL_RADIUS:
            reflect_ball(flipper_start, flipper_end)
            break

    # Überprüft Kollisionen mit den Spielfeldgrenzen
    if ball_pos[0] <= BALL_RADIUS or ball_pos[0] >= GAME_WIDTH - BALL_RADIUS:
        ball_vel[0] = -ball_vel[0]

    if ball_pos[1] <= BALL_RADIUS or ball_pos[1] >= HEIGHT - BALL_RADIUS:
        ball_vel[1] = -ball_vel[1]
```

Kollisionshandling an einem "passiven" Element

Diese Funktion behandelt Kollisionen der Kugel mit einem passiven Element (einem Bumper). Die Geschwindigkeit der Kugel wird reflektiert und modifiziert, um den Aufprall zu simulieren. Außerdem werden Particles bei einem Aufprall ausgelöst.

```
def reflect_ball_velocity(ball_pos, ball_vel, bumper_pos, bumper_radius):
    global ball_angular_vel

    # Berechnet den Einfallswinkel
    angle_of_incidence = math.atan2(ball_pos[1] - bumper_pos[1], ball_pos[0] -
bumper_pos[0])

    # Reflektiert den Geschwindigkeitsvektor
    normal = (math.cos(angle_of_incidence), math.sin(angle_of_incidence))
    dot_product = ball_vel[0] * normal[0] + ball_vel[1] * normal[1]
    ball_vel[0] -= 2 * dot_product * normal[0]
    ball_vel[1] -= 2 * dot_product * normal[1]

    # Multipliziert die Geschwindigkeit mit dem Bumper-Bounce-Faktor
    ball_vel[0] *= BUMPER_BOUNCE
    ball_vel[1] *= BUMPER_BOUNCE

    # Wendet Drehmoment basierend auf der Kollision an
    collision_vector = [ball_pos[0] - bumper_pos[0], ball_pos[1] -
bumper_pos[1]]
    torque = (collision_vector[0] * ball_vel[1] - collision_vector[1] *
ball_vel[0]) / (BALL_RADIUS ** 2)
    ball_angular_vel += torque

    # Stellt sicher, dass die Kugel nicht zu weit in den Bumper eindringt
    distance = math.hypot(ball_pos[0] - bumper_pos[0], ball_pos[1] -
bumper_pos[1])
    overlap = BALL_RADIUS + bumper_radius - distance
    if overlap > 0:
        ball_pos[0] += overlap * normal[0]
        ball_pos[1] += overlap * normal[1]

    # Fügt Partikel an der Kollisionsstelle hinzu
    add_particles(ball_pos)
```

Einstellbar: Position der Kugel

Diese Funktion ermöglicht es dem Benutzer, die Startposition der Kugel mit einem Mausklick festzulegen, solange das Spiel nicht gestartet ist. Das Spiel kann dann über die GUI gestartet werden.

```
def handle_mouse():
    global ball_pos
    if pygame.mouse.get_pressed()[0] and not GAME_STARTED:
        mouse_x, mouse_y = pygame.mouse.get_pos()

        if mouse_x < GAME_WIDTH and not
(initial_impulse_slider.get_abs_rect().collidepoint(mouse_x, mouse_y) or
gravity_strength_slider.get_abs_rect().collidepoint(mouse_x, mouse_y) or
launch_angle_slider.get_abs_rect().collidepoint(mouse_x, mouse_y)):
            ball_pos = list(pygame.mouse.get_pos())



Anzeige: Richtung und Geschwindigkeit der Kugel(n)
```

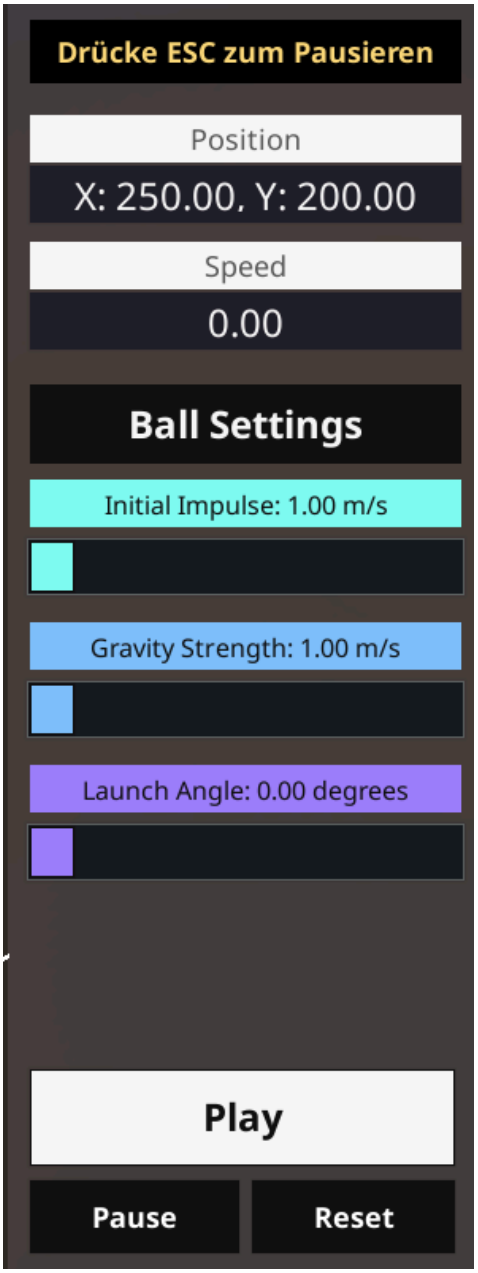
Anzeige: Richtung und Geschwindigkeit der Kugel(n)

Diese Funktionen zeichnen die Kugel und eine Linie, die die Richtung der Geschwindigkeit anzeigt. Außerdem wird die aktuelle Position und Geschwindigkeit der Kugel in der grafischen Benutzeroberfläche angezeigt.

```
def draw_ball():
    pygame.draw.circle(window, WHITE, (int(ball_pos[0]), int(ball_pos[1])),
BALL_RADIUS)
    direction_length = 30 * math.sqrt(ball_vel[0]**2 + ball_vel[1]**2) / 100
    angle = math.atan2(ball_vel[1], ball_vel[0])
    end_pos = (ball_pos[0] + direction_length * math.cos(angle), ball_pos[1] +
direction_length * math.sin(angle))
    pygame.draw.line(window, PURPLE, (ball_pos[0], ball_pos[1]), end_pos, 2)

def draw_gui():
    position_text = f'X: {ball_pos[0]:.2f}, Y: {ball_pos[1]:.2f}'
    speed = math.sqrt(ball_vel[0]**2 + ball_vel[1]**2) / 100
    speed_text = f'{speed:.2f}'

    position_value.set_text(position_text)
    speed_value.set_text(speed_text)

position_label = UILabel(
    relative_rect=pygame.Rect((GAME_WIDTH + 14, 70), (UI_WIDTH - 28, 30)),
    text="Position",
    manager=manager,
    object_id=ObjectID(class_id='@label', object_id='#position_label')
)

position_value = UITextBox(
    relative_rect=pygame.Rect((GAME_WIDTH + 12, 100), (UI_WIDTH - 24, 40)),
    html_text="X: 0.00, Y: 0.00",
    manager=manager,
    object_id=ObjectID(class_id='@text_box', object_id='#position_value')
)

speed_label = UILabel(
    relative_rect=pygame.Rect((GAME_WIDTH + 14, 150), (UI_WIDTH - 28, 30)),
    text="Speed",
    manager=manager,
    object_id=ObjectID(class_id='@label', object_id='#speed_label')
)

speed_value = UITextBox(
    relative_rect=pygame.Rect((GAME_WIDTH + 12, 180), (UI_WIDTH - 24, 40)),
    html_text="0.00",
    manager=manager,
    object_id=ObjectID(class_id='@text_box', object_id='#speed_value')
)
```

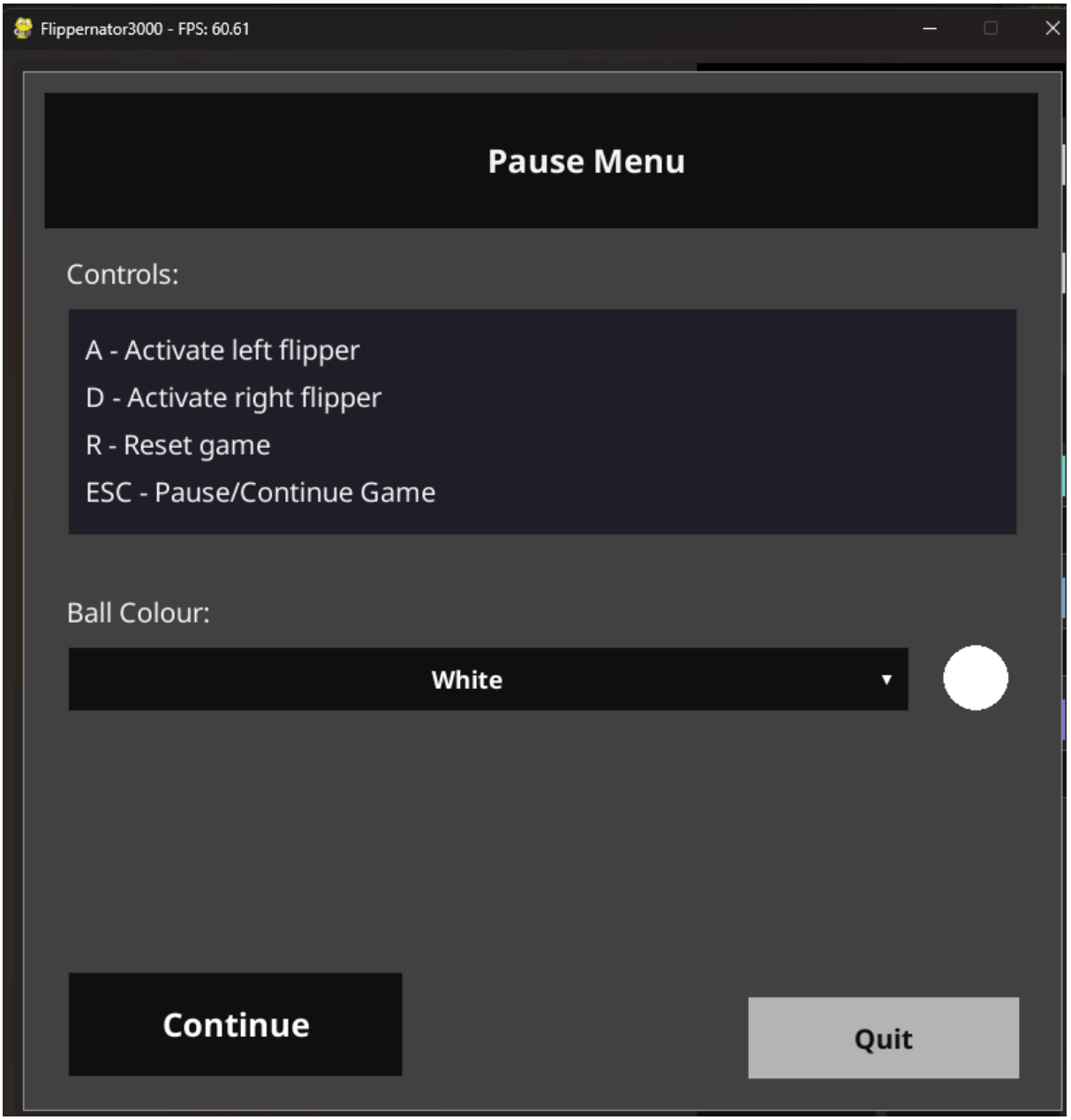
GUI

Game Settings:



- Anzeige der aktuellen Ball-Position
- Anzeige der aktuellen Ball-Geschwindigkeit
- Anpassbarer Slider für den Startimpuls
- Anpassbarer Slider für die Gravitationsstärke
- Anpassbarer Slider für den Startwinkel
- Buttons für Spielstart, Aufrufen des Pause Menüs, Spiel zurücksetzen

Pause Menu:



- Liste der Steuerung
- Anpassung der Farbe des Balles
- Buttons zum Fortsetzen und Anwendung schließen

Der relevante Code für das GUI ist ungefähr von Zeile 500 -83, sowie teilweise in der game_loop-Funktion. Design der UI-Elemente sind in data/theme.json zu finden.

Ausführbare Datei

Die ausführbare Datei (.exe) finden Sie im Ordner “Flippernator3000 Game”, darin ist die Datei “Flippernator3000.exe”.

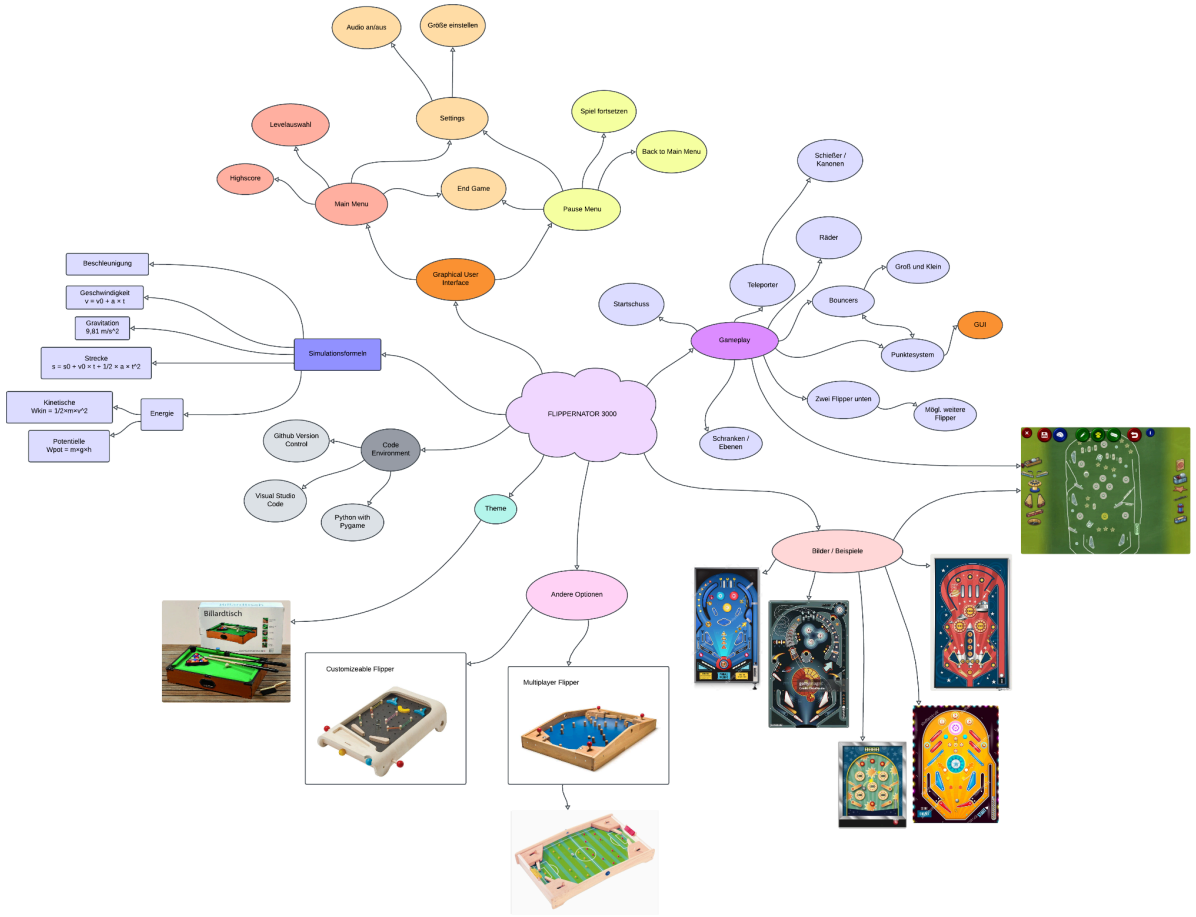
Steuerung

A - Flipper Links aktivieren
D - Flipper Rechts aktivieren
R - Spiel zurücksetzen
ESC - Spiel pausieren / fortsetzen

Information zum Spielen: Es muss auf dem Spielfeld eine Position der Kugel mit der Maus gewählt werden, dann lässt sich das Spiel erst starten.

Abgabe 1

Brainstorming (Lucidspark)



Code Snippets

1. Die Kugel bewegt sich in einer beliebigen Richtung:
Das folgende Snippet aus der Funktion move_ball zeigt, wie die Bewegung der Kugel basierend auf ihrer aktuellen Geschwindigkeit aktualisiert wird. Dies ermöglicht es der Kugel, sich in jede Richtung zu bewegen, abhängig von ihrer Geschwindigkeitsvektor-Komponenten ball_vel[0] und ball_vel[1]:

```
# main.py

# Aktualisiere die horizontale / vertikale Position der Kugel
ball_pos[0] += ball_vel[0]
ball_pos[1] += ball_vel[1]
```

2. Eine oder mehrere Beschleunigungen können wirken:
Das Snippet zeigt, wie die Schwerkraft als eine konstante Beschleunigung auf die Kugel wirkt, wodurch ihre vertikale Geschwindigkeit kontinuierlich angepasst wird. Der INITIAL_BALL_IMPULSE kann mit einem Slider zwischen 1 und 10 festgelegt werden (Default = 1) und gibt der Kugel eine Anfangsbeschleunigung:

```
# config.py

INITIAL_BALL_IMPULSE = 1
```

```
# main.py

if ball_vel == [0, 0]:
    # Setze den Anfangsimpuls nach unten
    ball_vel[1] = INITIAL_BALL_IMPULSE

# Schwerkraft anwenden, die die Kugel nach unten zieht
ball_vel[1] += GRAVITY
```

3. Startimpuls mit einer frei einstellbaren Richtung und Stärke &
4. Einstellbar: Startposition, Startimpuls, Beschleunigungseinflüsse:
Das folgende Snippet aus handle_mouse zeigt, wie das Spiel durch einen Mausklick gestartet wird. Dabei wird der Startpunkt der Kugel genau da hin

gesetzt, auf den man im Game-Window drückt. Man hat davor noch die Option, mit Hilfe von Slidern, die Richtung, Stärke und die Gravitationsstärke einzustellen.

```
# main.py

def handle_mouse():
  global ball_pos, ball_vel, GAME_STARTED, BALL_ANGLE, INITIAL_BALL_IMPULSE,
GRAVITY_STRENGTH, GRAVITY
  if pygame.mouse.get_pressed()[0]:
    mouse_x, mouse_y = pygame.mouse.get_pos()
    # Überprüfe, ob die Maus auf einem der Slider ist
    if slider1_rect.collidepoint(mouse_x, mouse_y):
      INITIAL_BALL_IMPULSE = int((mouse_x - slider1_rect.left) /
slider1_rect.width * (SLIDER_MAX_VALUE - SLIDER_MIN_VALUE) + SLIDER_MIN_VALUE)
    elif slider2_rect.collidepoint(mouse_x, mouse_y):
      GRAVITY_STRENGTH = int((mouse_x - slider2_rect.left) / slider2_rect.width
* (SLIDER_MAX_VALUE - SLIDER_MIN_VALUE) + SLIDER_MIN_VALUE)
      GRAVITY = 0.1 * GRAVITY_STRENGTH
    elif angle_slider_rect.collidepoint(mouse_x, mouse_y):
       BALL_ANGLE = int((mouse_x - angle_slider_rect.left) /
angle_slider_rect.width * (SLIDER_MAX_ANGLE - SLIDER_MIN_ANGLE) +
SLIDER_MIN_ANGLE)

  else:
  # Starte das Spiel nur, wenn außerhalb der Sliderbereiche geklickt wird
    if not GAME_STARTED:
      angle_rad = math.radians(BALL_ANGLE + 90)
      ball_vel = [
        INITIAL_BALL_IMPULSE * math.cos(angle_rad),
        INITIAL_BALL_IMPULSE * math.sin(angle_rad)
      ]
      ball_pos = list(pygame.mouse.get_pos())
      GAME_STARTED = True
```

Weitere Wichtige Methoden:

Bewegung der Kugel

In der Funktion move_ball werden alle Auswirkungen auf das Ball-Movement angewendet. Dabei werden Einflüsse wie GRAVITY, DAMPENING oder INITIAL_BALL_IMPULS übertragen und die Geschwindigkeit der Kugel aktualisiert. Auch Kollisionen mit Außenwänden und Bumpern werden hier berechnet.

```
def move_ball():
    global GRAVITY, INITIAL_BALL_IMPULSE, BUMPER_BOUNCE

    # Wenn das Spiel nicht gestartet ist, wird die Funktion vorzeitig verlassen.
    if not GAME_STARTED:
        return

    # Wenn die Kugel noch keine Anfangsgeschwindigkeit hat
    if ball_vel == [0, 0]:
        ball_vel[1] = INITIAL_BALL_IMPULSE  # Setze den Anfangsimpuls nach unten

    # Schwerkraft anwenden, die die Kugel nach unten zieht
    ball_vel[1] += GRAVITY * DAMPENING

    # Aktualisiere die horizontale / vertikale Position der Kugel
    ball_pos[0] += ball_vel[0]
    ball_pos[1] += ball_vel[1]

    # Überprüfung auf Kollision mit den Seitenwänden des Spielfelds
    if ball_pos[0] <= BALL_RADIUS or ball_pos[0] >= WIDTH - BALL_RADIUS:
        # Kehre die horizontale Geschwindigkeit um
        ball_vel[0] = -ball_vel[0]

    if ball_pos[1] <= BALL_RADIUS or ball_pos[1] >= HEIGHT - BALL_RADIUS:
        # Kehre die vertikale Geschwindigkeit um
        ball_vel[1] = -ball_vel[1]

    # Kollision mit Bumpern
    for bumper in bumpers:
        if math.hypot(ball_pos[0] - bumper['pos'][0], ball_pos[1] -
bumper['pos'][1]) < BALL_RADIUS + bumper['radius']:
            if not bumper['active']:
                bumper['active'] = True
                bumper['timer'] = 10  # Anzahl der Frames, die die Animation
dauert
            angle = math.atan2(ball_pos[1] - bumper['pos'][1], ball_pos[0] -
bumper['pos'][0])
            ball_vel[0] += BUMPER_BOUNCE * math.cos(angle)
            ball_vel[1] += BUMPER_BOUNCE * math.sin(angle)
```

Kollision der Kugel mit den Flippern

Damit das Spiel auch spielbar ist, haben die Flipper Kollisionen. Bei den Flippern wird außerdem ein FLIPPER_BOUNCE hinzugefügt, damit das Spiel nicht sofort zu Ende ist.

```
def reflect_ball(start, end):
  global FLIPPER_BOUNCE, DAMPENING

  normal = get_line_normal(start, end)
  midpoint = ((start[0] + end[0]) / 2, (start[1] + end[1]) / 2)
  ball_to_midpoint = (midpoint[0] - ball_pos[0], midpoint[1] - ball_pos[1])

  if (ball_to_midpoint[0] * normal[0] + ball_to_midpoint[1] * normal[1]) > 0:
    # Normalenvektor umkehren, wenn er zum Flipper zeigt
    normal = (-normal[0], -normal[1])

  new_velocity = reflect((ball_vel[0], ball_vel[1]), normal)

  ball_vel[0] = (new_velocity[0] + abs(normal[0]) *- FLIPPER_BOUNCE)
  ball_vel[1] = (new_velocity[1] + abs(normal[1]) *- FLIPPER_BOUNCE)

def check_collision():
  # Überprüft Kollisionen zwischen der Kugel und den Flippern und handhabt die
Folgen einer Kollision
  global ball_pos, ball_vel, collision_cooldown

  if collision_cooldown > 0:
    collision_cooldown -= 1
    return

  for flipper_pos, angle, is_right in [(left_flipper_pos, left_flipper_angle,
False), (right_flipper_pos, right_flipper_angle, True)]:
    # Berechne die Positionen der Außenwände des Flippers
    start_x, start_y = flipper_pos
    end_x = start_x + FLIPPER_LENGTH * math.cos(math.radians(angle)) * (-1 if
is_right else 1)
    end_y = start_y - FLIPPER_LENGTH * math.sin(math.radians(angle))
    normal = get_line_normal((start_x, start_y), (end_x, end_y))
    perpendicular = (-normal[1], normal[0])

    # Überprüfe Kollision mit den Außenwänden des Flippers
    wall_start = (start_x + perpendicular[0] * FLIPPER_WIDTH / 2, start_y +
perpendicular[1] * FLIPPER_WIDTH / 2)
    wall_end = (end_x + perpendicular[0] * FLIPPER_WIDTH / 2, end_y +
perpendicular[1] * FLIPPER_WIDTH / 2)
    if point_line_distance(ball_pos, wall_start, wall_end) <= BALL_RADIUS:
       reflect_ball(wall_start, wall_end)
       collision_cooldown = COLLISION_COOLDOWN_MAX
       break
```

GUI

Im GUI werden folgende Informationen dargestellt:

- Hinweis zum Pause-Menü
- Aktuelle Position der Kugel auf 2 Nachkommastellen gerundet
- Aktuelle Geschwindigkeit der Kugel auf 2 Nachkommastellen gerundet
- Slider für Startgeschwindigkeit, Startwinkel und Stärke der Gravitation

```
def draw_gui():
    # Zeigt die GUI-Elemente auf dem Bildschirm an, einschließlich der aktuellen
Position und Geschwindigkeit der Kugel.
    position_text = f'X: {ball_pos[0]:.2f}, Y: {ball_pos[1]:.2f}'

    speed = math.sqrt(ball_vel[0]**2 + ball_vel[1]**2)
    speed_text = f'Speed: {speed:.2f}'

    pause_text = "Drücke ESC zum Pausieren"

    position_surf = font.render(position_text, True, pygame.Color('white'))
    speed_surf = font.render(speed_text, True, pygame.Color('white'))
    pause_surf = font.render(pause_text, True, pygame.Color('yellow'))

    window.blit(pause_surf, (10, 10))
    window.blit(position_surf, (10, 40))
    window.blit(speed_surf, (10, 70))

def draw_slider(slider_rect, slider_value, text, text_pos, min_value,
max_value):
    pygame.draw.rect(window, SLIDER_COLOR, slider_rect)
    # Berechne die Position des Handles basierend auf dem Slider-Wert
    normalized_value = (slider_value - min_value) / (max_value - min_value)
    handle_pos = slider_rect.left + normalized_value * (slider_rect.width -
SLIDER_HEIGHT)
    handle_rect = pygame.Rect(handle_pos, slider_rect.centery - SLIDER_HEIGHT //
2, SLIDER_HEIGHT, SLIDER_HEIGHT)
    pygame.draw.rect(window, SLIDER_HANDLE_COLOR, handle_rect)
    text_with_value = f"{text}: {slider_value}"
    text_surf = font.render(text_with_value, True, SLIDER_TEXT_COLOR)
    window.blit(text_surf, text_pos)

# Im Gameloop werden die Slider dann gezeichnet
draw_slider(slider1_rect, INITIAL_BALL_IMPULSE, "Initial Ball Impulse", (320,
20), SLIDER_MIN_VALUE, SLIDER_MAX_VALUE)

draw_slider(slider2_rect, GRAVITY_STRENGTH, "Gravity Strength", (320, 70),
SLIDER_MIN_VALUE, SLIDER_MAX_VALUE)

draw_slider(angle_slider_rect, BALL_ANGLE, "Launch Angle", (320, 120),
SLIDER_MIN_ANGLE, SLIDER_MAX_ANGLE)
```