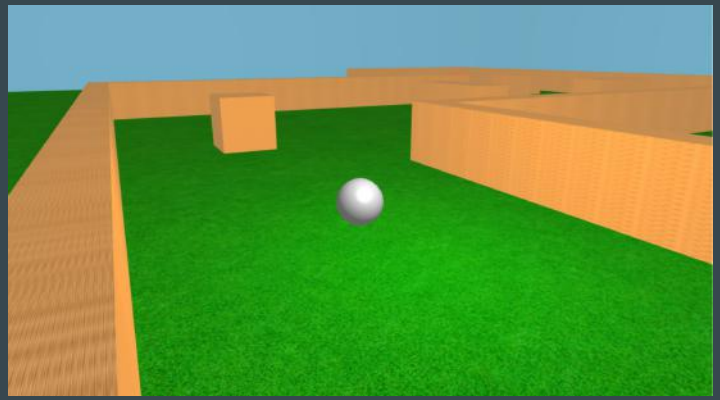
Golf For It

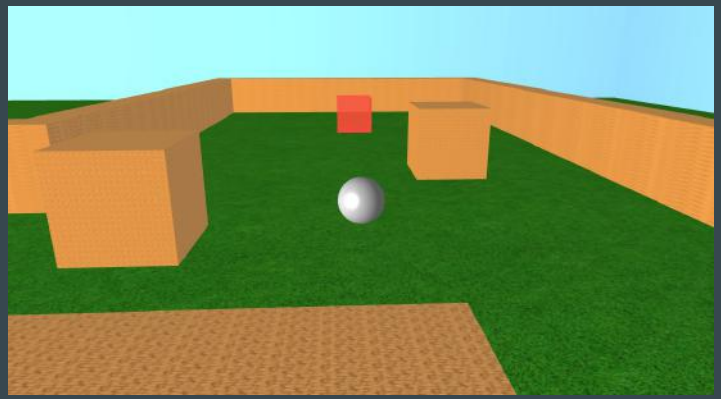
...

Group 8:
Charles Zhang, Clement Nguyen,
Isabelle Marchand, Victoria Delk

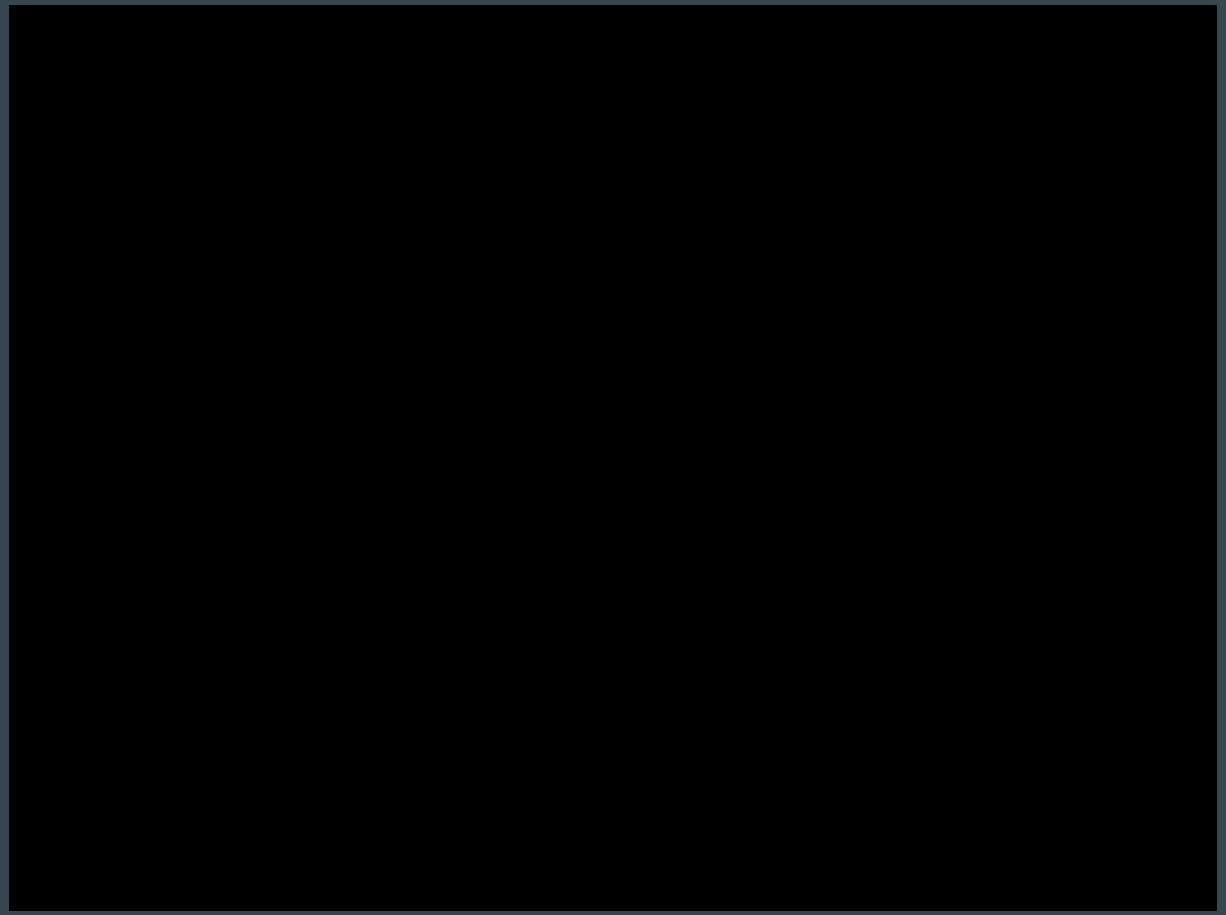
Game Overview

- Mini-golf simulation game in third-person perspective of golf ball
- Objective: hit goal in as few strokes as possible while navigating around obstacles
- Game is reset automatically after reaching goal



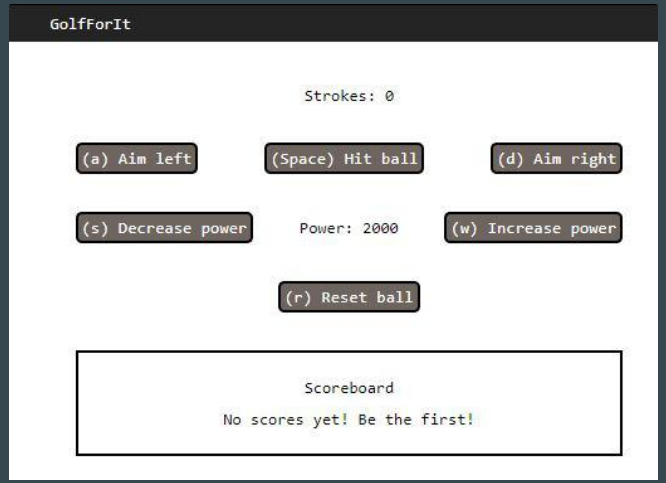


Demo



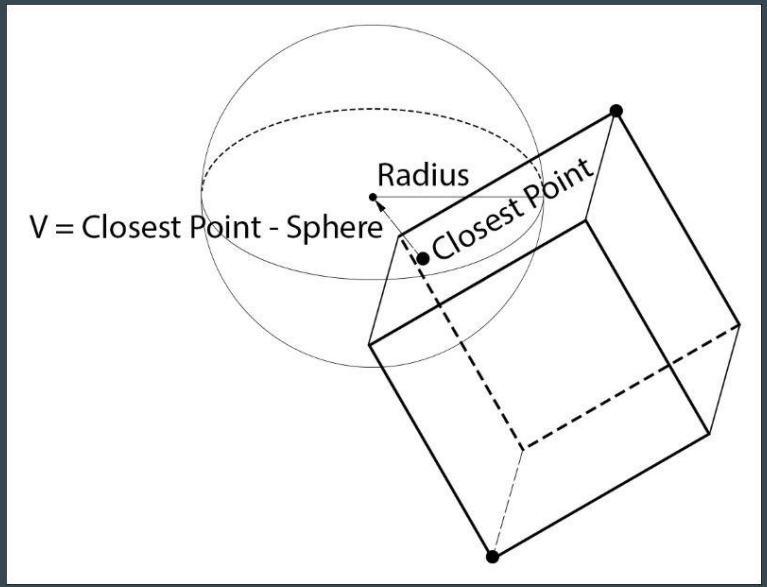
Control Panel

- Hitting and Aiming Controls
  - w and s to increase/decrease power
  - a and d to aim left and right
- Stroke Counter
  - Counts number of strokes user has taken
- Scoreboard
  - Tracks 3 highest scores
  - Updates when player reaches the goal



Collision Detection

- Determines if a box collider and a sphere collider are colliding with each other
- Find the point on the box closest to the sphere
- If the closest point is within the radius of the sphere, the box and sphere are colliding
- Only works with axis-aligned bounding box

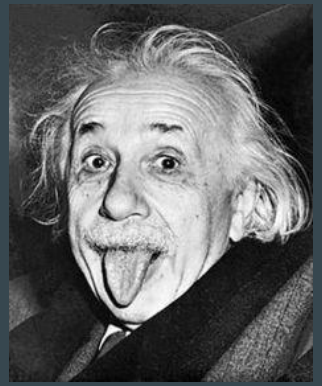


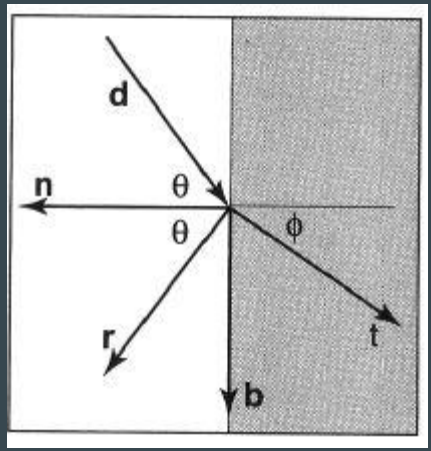
Collision Detection

```
34  function test_box_sphere_collision(box, sphere) {
35      let pos_delta = sphere.transform.position.minus(box.transform.position);
36      let closest_point_on_aabb = vec3(
37          clamp(pos_delta[0], -box.transform.size[0], box.transform.size[0]),
38          clamp(pos_delta[1], -box.transform.size[1], box.transform.size[1]),
39          clamp(pos_delta[2], -box.transform.size[2], box.transform.size[2])
40      );
41      let collision_delta = pos_delta.minus(closest_point_on_aabb);
42      let distance = collision_delta.norm();
43
44      if (distance < sphere.transform.size[0]) {
45          return collision_delta.normalized();
46      }
47
48      return null;
49  }
```

Physics

- Rotational Velocity
  - Rotational velocity is applied to the ball when it is hit by the player
  - This rotational velocity is used to calculate the position of the ball
  - Drag is applied to slow down the ball's rotational velocity over time
  - When the rotational velocity drops below a certain threshold, it's clamped to 0
- Bouncing
  - On collision, the ball bounces off the wall
  - The trajectory of the bounce is determined by the ball's directional velocity and the normal of the wall





Physics

```
const applyFriction = (ball) => {
    const dx = frictionForce * Math.abs(Math.sin(ball.transform.rotation[1]));
    const dz = frictionForce * Math.abs(Math.cos(ball.transform.rotation[1]));
    // Apply x-friction if the ball is in motion in the x-direction
    if (ball.rigidbody.velocity[0] !== 0) {
        if (ball.rigidbody.velocity[0] > 0) {
            ball.rigidbody.velocity[0] -= dx;
        } else {
            ball.rigidbody.velocity[0] += dx;
        }
    }
    // Apply z-friction if the ball is in motion in the z-direction
    if (ball.rigidbody.velocity[2] !== 0) {
        if (ball.rigidbody.velocity[2] > 0) {
            ball.rigidbody.velocity[2] -= dz;
        } else {
            ball.rigidbody.velocity[2] += dz;
        }
    }
    // Clamp velocity to 0
    if (ball.rigidbody.velocity.norm() < epsilon ** 2) {
        ball.rigidbody.velocity[0] = 0;
        ball.rigidbody.velocity[2] = 0;
    }
};
```

Physics

```
45  const calculateBounce = (ball, normal) => {
46      // Calc new velocity
47      const v = ball.rigidbody.velocity;
48      ball.rigidbody.velocity = v.minus(normal.times(2 * v.dot(normal)));
49      // Calc new angle
50      const vp = ball.rigidbody.velocity;
51      const dot = v[0] * normal[0] + v[0] * normal[2];
52      const det = v[0] * normal[2] - v[2] * normal[0];
53      const angle = Math.atan2(det, dot);
54      if (angle > 0) {
55          ball.transform.rotation[1] -= Math.acos(
56              v.dot(vp) / (v.norm() * vp.norm())
57          );
58      } else {
59          ball.transform.rotation[1] += Math.acos(
60              v.dot(vp) / (v.norm() * vp.norm())
61          );
62      }
63  };
```

Rendering

- Each game object contains a position, rotation, size, shape, and material
- The render_game_object function applies affine matrix transformations on the identity matrix based on the game object's position, rotation, and size
- Then, it draws the game object using its shape and material

```
function render_game_object(context, program_state, game_object) {
    if (game_object.is_enabled && game_object.renderer.is_enabled) {
        let model_transform = Mat4.identity()
            .times(Mat4.translation(
                game_object.transform.position[0],
                game_object.transform.position[1],
                game_object.transform.position[2]
            ))
            .times(Mat4.rotation(game_object.transform.rotation[0], 1, 0, 0))
            .times(Mat4.rotation(game_object.transform.rotation[1], 0, 1, 0))
            .times(Mat4.rotation(game_object.transform.rotation[2], 0, 0, 1))
            .times(
                Mat4.scale(
                    game_object.transform.size[0],
                    game_object.transform.size[1],
                    game_object.transform.size[2]
                )
            );
        game_object.renderer.shape.draw(
            context,
            program_state,
            model_transform,
            game_object.renderer.material
        );
    }
}
```

Improvements

- Add more collision detection types
  - Box vs. box
  - Sphere vs. sphere
  - Non-aligned boxes
- Add mouse controls for aiming
- Extended levels and courses
- Add texture to sky and golf ball
- More obstacle variety
- Implement gravity/y-axis movement

The End / Questions

Thank You :)