Piłką na platformach i przez teleporty

Andrzej P.Urbański
Wydział Informatyki
Politechnika Poznańska

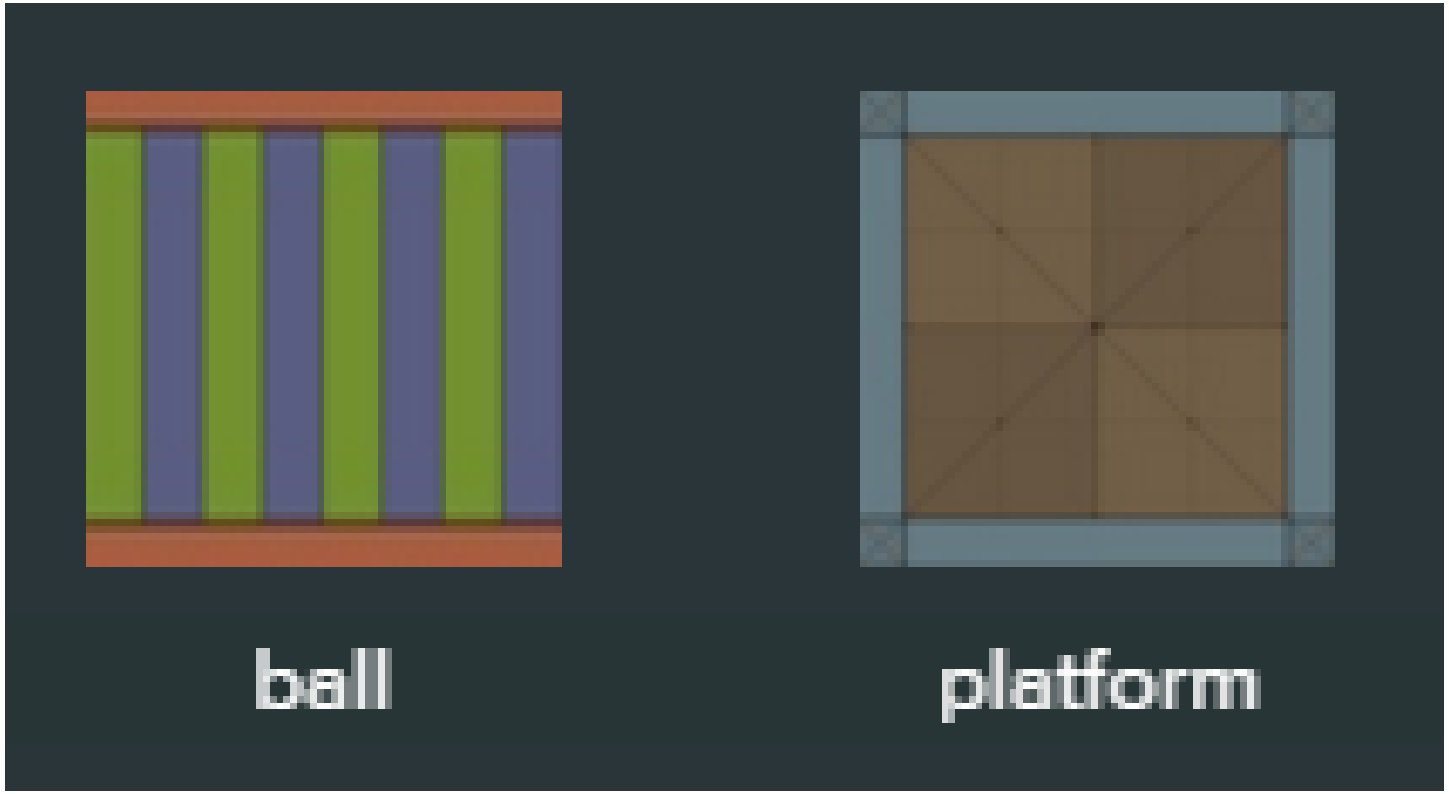
Play Canvas – system programowania gier

- Zintegrowane narzędzia do tworzenia obiektów 3D, skryptów JavaScript i programów GLCS
- Wbudowane modelowanie fizyki ciał rzeczywistych: sprężystość, uginanie, oddziaływanie grawitacji

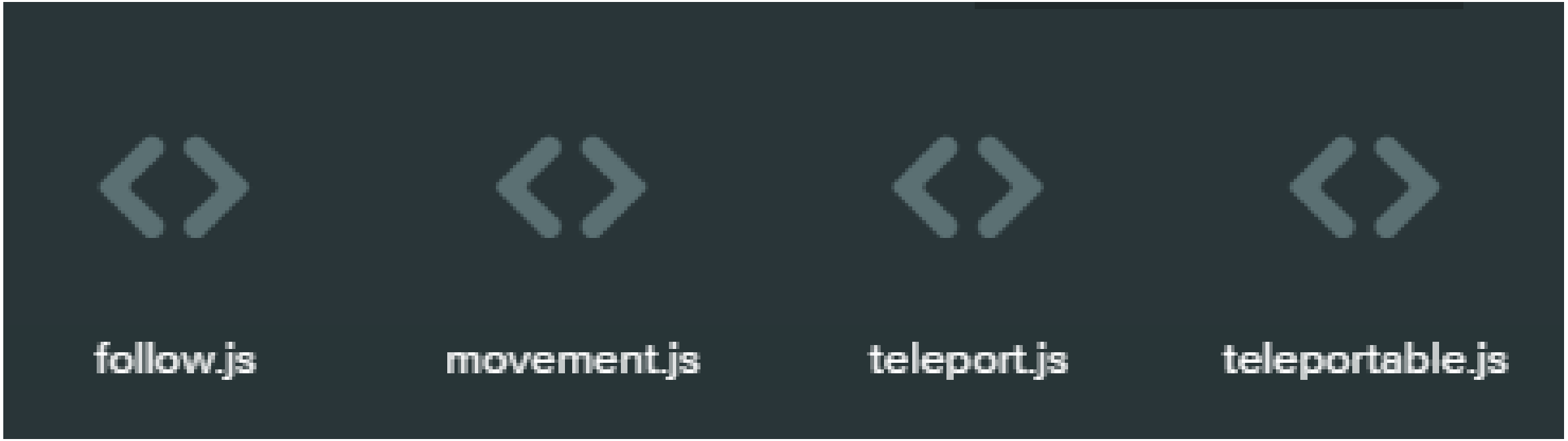
Modele – kształty obiektów

Materiały - sprężystość

Tekstury – wzory graficzne



Skrypty – JavaSript z API PC



Obiektowość

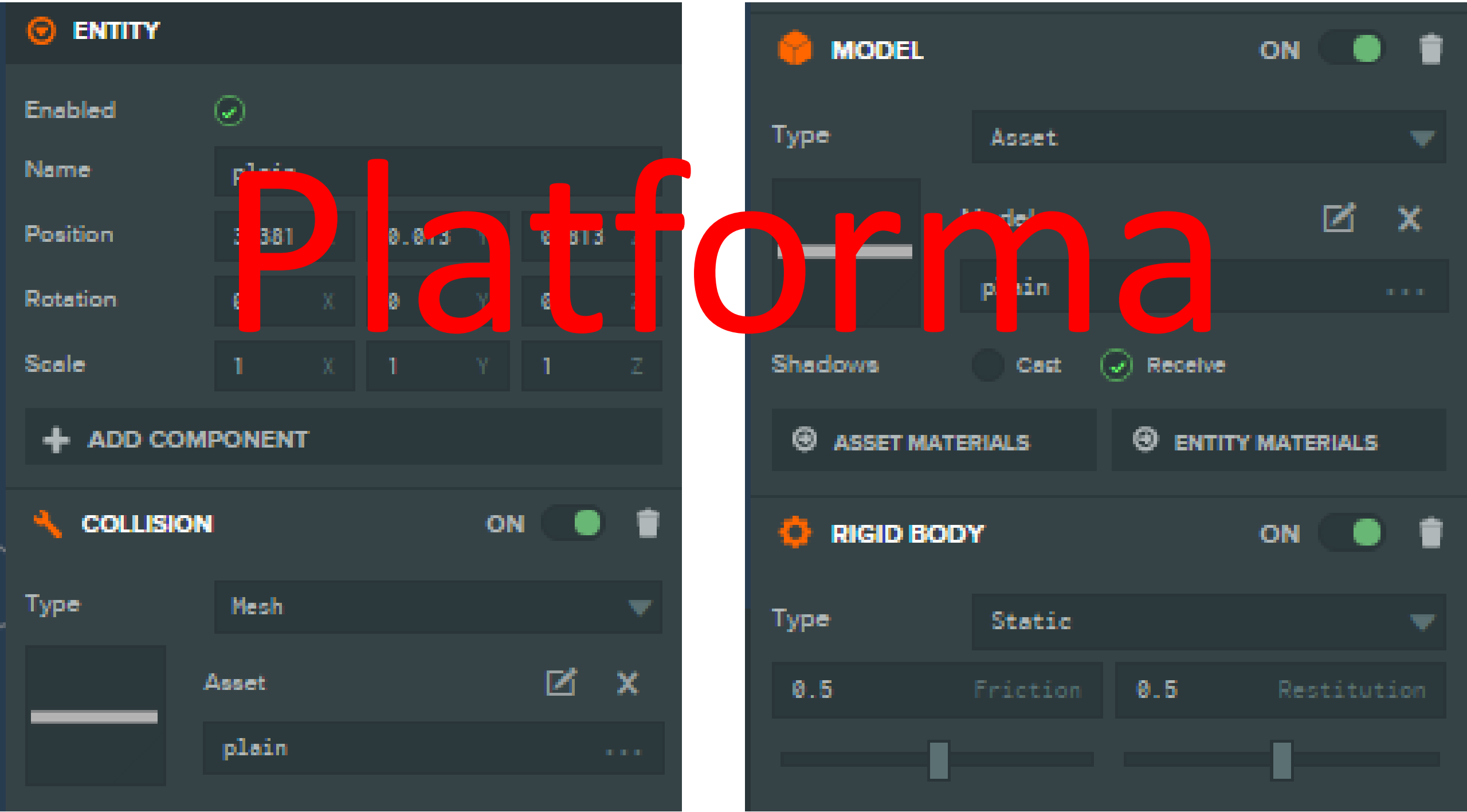
- Kształt obiektu
- Wygląd zewnętrzny
- Interakcja z innymi obiektami (collision)
- Siła ciążenia i bezwładności (rigid body=dynamic)
- Skrypty nadające indywidualne cechy

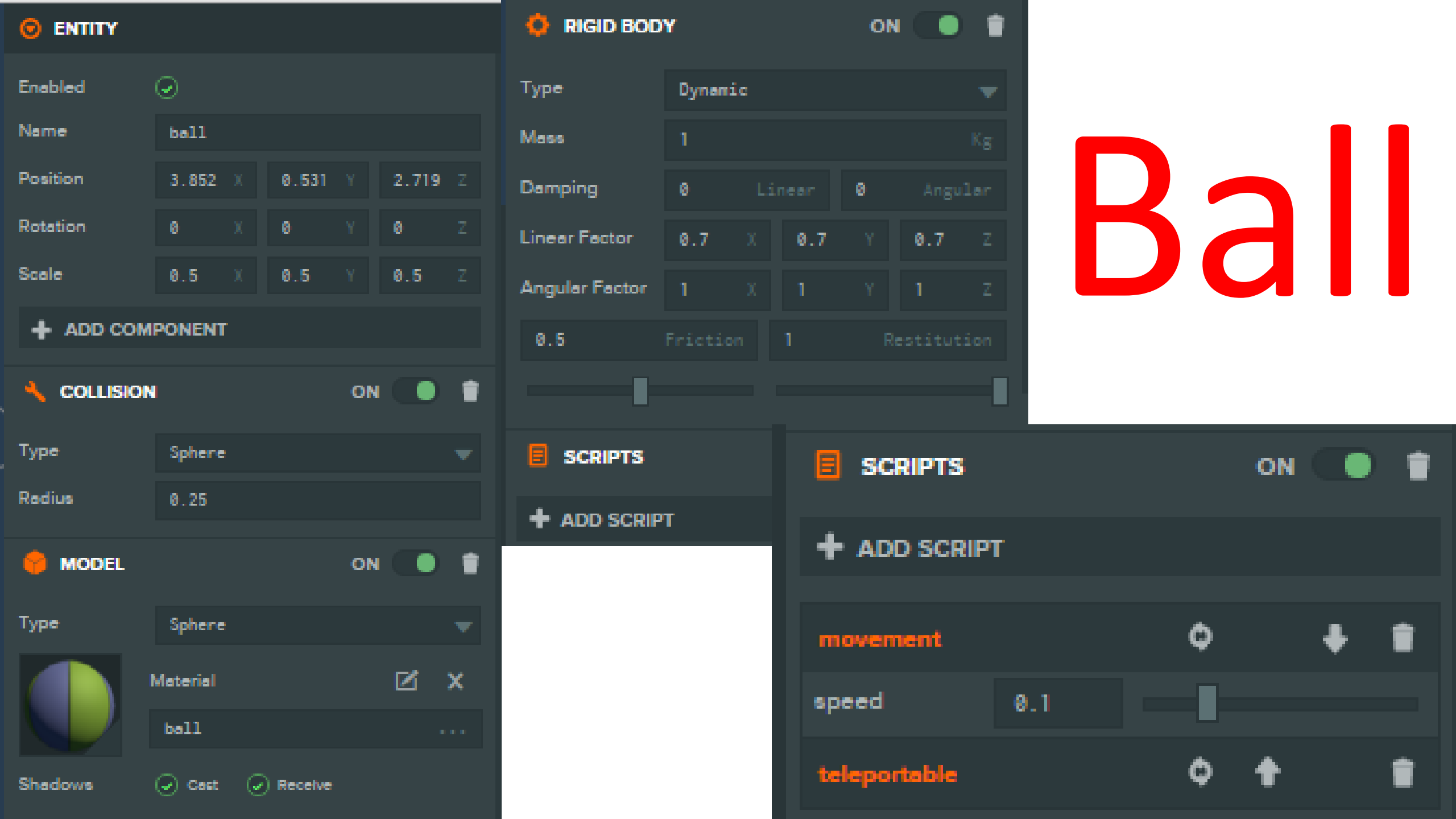
Atrybuty

- Zarówno predefiniowane obiekty jak skrypty mają atrybuty, które można ustawiać w trakcie dodawania obiektu
- W definicji skryptu można deklarować atrybuty, które ujawnią się po przypisaniu skryptu do obiektu

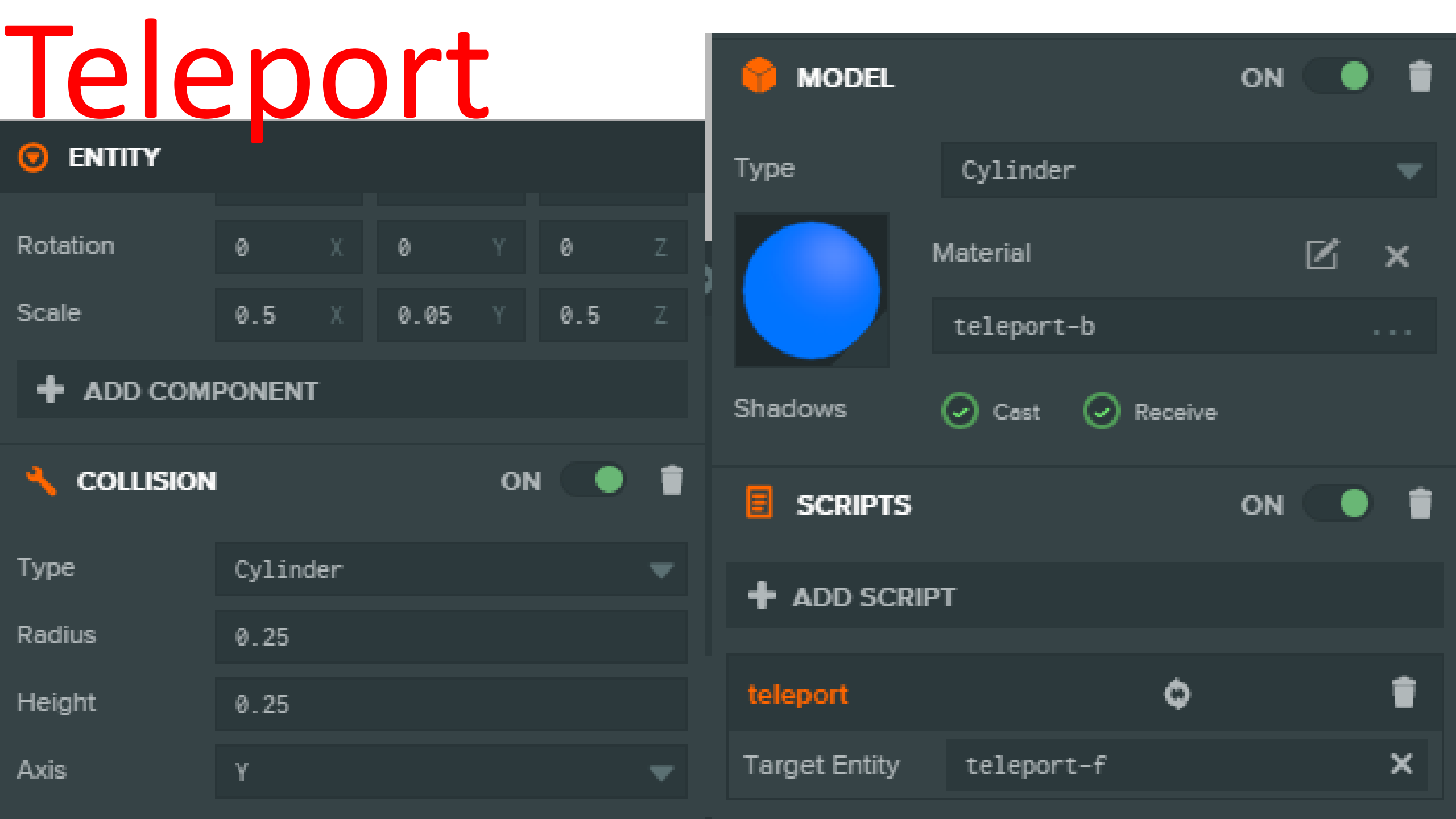
Obiekty zabawy z piłką

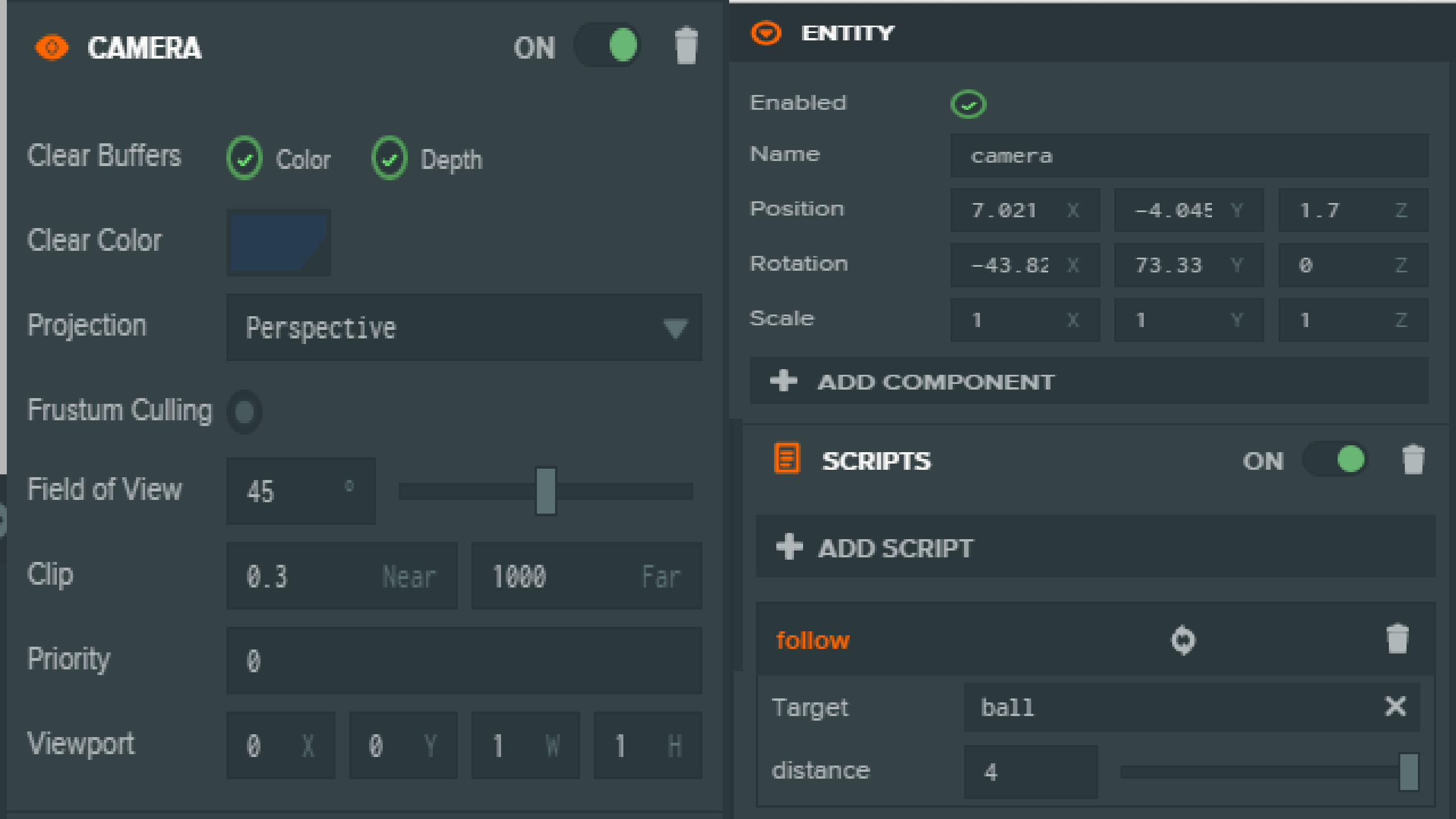
- Platformy, po których można kulać piłkę
- Piłka
- Teleport, który przenosi piłkę
- Kamera, która sterowana skryptem dokładnie pokazuje piłkę i najbliższe otoczenie

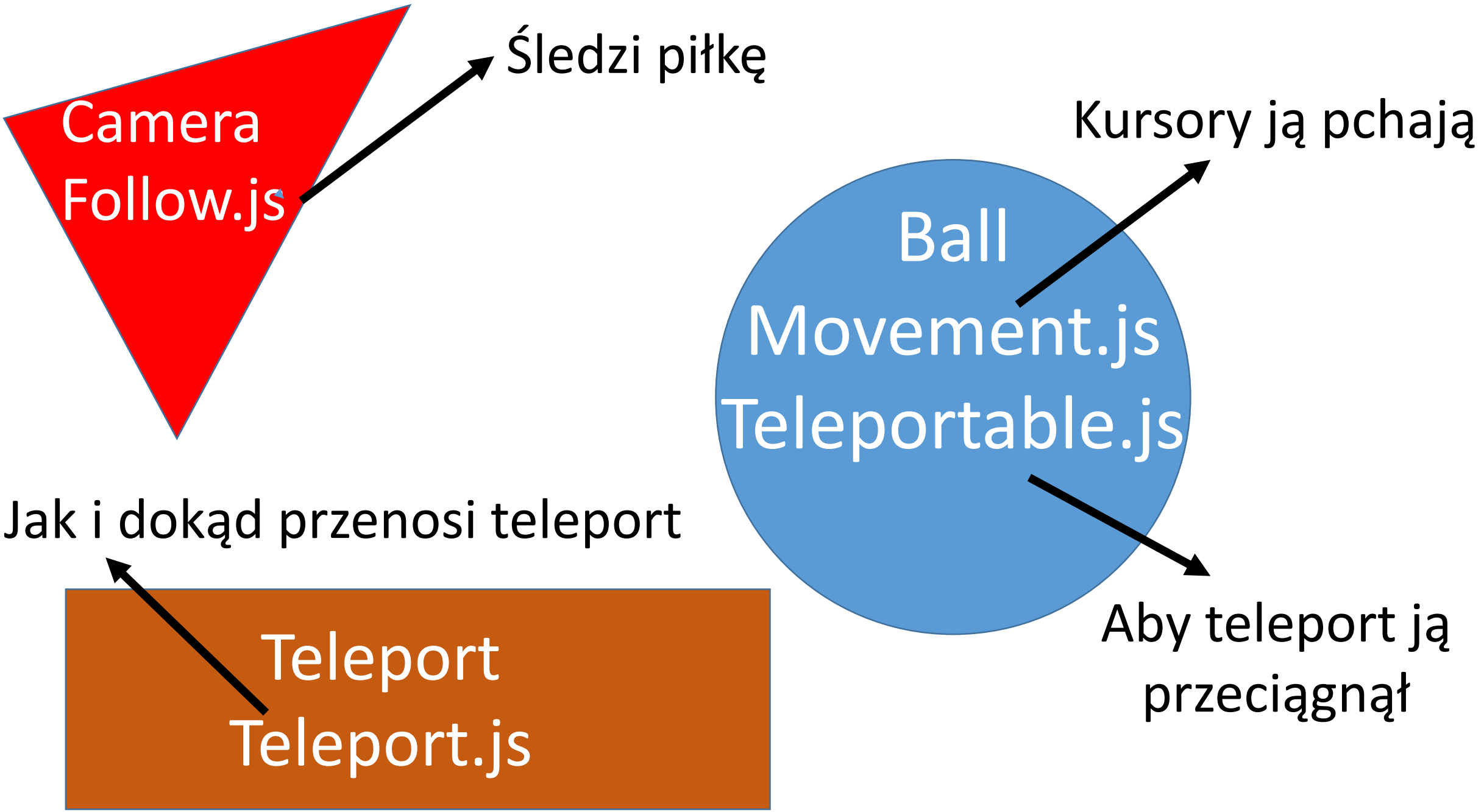

ENTITY
Enabled
Name
Position
Rotation
Scale
1
X
1
Y
1
Z
ADD COMPONENT
COLLISION
ON
Type
Mesh
Asset
plain
MODEL
ON
Type
Asset
Shadows
Cast
Receive
ASSET MATERIALS
ENTITY MATERIALS
RIGID BODY
ON
Type
Static
0.5
Friction
0.5
Restitution
Platforma


ENTITY
Enabled
Name
ball
Position
3.852 X
0.531 Y
2.719 Z
Rotation
0 X
0 Y
0 Z
Scale
0.5 X
0.5 Y
0.5 Z
ADD COMPONENT
COLLISION
ON
Type
Sphere
Radius
0.25
MODEL
ON
Type
Sphere
Material
ball
Shadows
Cast
Receive
RIGID BODY
ON
Type
Dynamic
Mass
1
Kg
Damping
0
Linear
0
Angular
Linear Factor
0.7 X
0.7 Y
0.7 Z
Angular Factor
1 X
1 Y
1 Z
0.5
Friction
1
Restitution
SCRIPTS
ADD SCRIPT
SCRIPTS
ON
ADD SCRIPT
movement
speed
0.1
teleportable
Ball

Teleport


ENTITY
Rotation
0 X
0 Y
0 Z
Scale
0.5 X
0.05 Y
0.5 Z
ADD COMPONENT
COLLISION
ON
Type
Cylinder
Radius
0.25
Height
0.25
Axis
Y
MODEL
ON
Type
Cylinder
Material
teleport-b
Shadows
Cast
Receive
SCRIPTS
ON
ADD SCRIPT
teleport
Target Entity
teleport-f


CAMERA
ON
Clear Buffers
Color
Depth
Clear Color
Projection
Perspective
Frustum Culling
Field of View
45
Clip
0.3
Near
1000
Far
Priority
0
Viewport
0 X
0 Y
1 W
1 H
ENTITY
Enabled
Name
camera
Position
7.021 X
-4.045 Y
1.7 Z
Rotation
-43.82 X
73.33 Y
0 Z
Scale
1 X
1 Y
1 Z
ADD COMPONENT
SCRIPTS
ON
ADD SCRIPT
follow
Target
ball
distance
4


Camera
Follow.js
Śledzi piłkę
Ball
Movement.js
Teleportable.js
Kursory ją pchają
Aby teleport ją
przeciągnął
Jak i dokąd przenosi teleport
Teleport
Teleport.js

- W każdej grze kamera śledzi miejsce akcji
- W każdej grze jest obiekt, z którym gracz może się utożsamić, bo wprawia go w ruch kursorami bądź myszką
- Niezwykłość teleportów – przenoszenie ludzi i przedmiotów realizuje specjalnie opracowany skrypt „teleport.js”, który wie w jakie miejsce ma nastąpić przeniesienie

Budowa przestrzeni gry 1

- Przestrzeń gry można budować bez ingerowania w skrypty
- Zestawiając platformy
- Tworząc teleport
  - Przez z duplikowanie niebeiskiego lub zielonego albo
  - Tworząc cylinder – kilknąć „+” w hierarchy „add cylinder”
  - Ustawić skalę x=0.5, y=0.05, z=0.5
  - Wstawić w Model material teleport-a lub teleport-b
  - W przypadku teleportu niebieskiego(wejściowego) dołączyć skrypt „teleport.js” z atrybutem wskazującym teleport wyjściowy dla niego

Budowa przestrzeni gry 2

- Ustawiając odpowiednio teleporty:
  - niebieski jako wynoszący,
  - zielony jako przynoszący
- Każdy teleport powinien mieć unikalną nazwę
- W niebieskim teleporcie koniecznie trzeba ustawić atrybut skryptu „Target entity” na teleport, do którego nastąpi przeniesienie

Budowa skryptu PC: atrybuty, tworzenie, działanie

```
pc.script.attribute(,nazwa', 'number',1,{min:0,max:5,step:0.5,decimalPrecision:2});
pc.script.create(,nazwa', function (app) {
var nazwa= function (entity) {
        this.entity = entity;
        this.force = new pc.Vec3();
    };
    nazwa.prototype = {
    …
    return nazwa;
});
```

Teleport

```
// The target entity where we are going to teleport
pc.script.attribute('target', 'entity', null, {
    displayName: 'Target Entity'
});

pc.script.create('teleport', function (app) {
    var Teleport = function (entity) {
        this.entity = entity;
    };

    Teleport.prototype = {
        initialize: function () {
            if (this.target) {
                // Subscribe to the triggerenter event of this entity's collision component.
                // This will be fired when a rigid body enters this collision volume.
                this.entity.collision.on('triggerenter', this.onTriggerEnter, this);
            }
        },
        onTriggerEnter: function (otherEntity) {
            // it is not teleportable
            if (! otherEntity.script.teleportable)
                return;

            // teleport entity to the target entity
            otherEntity.script.teleportable.teleport(this.entity, this.target);
        }
    };

    return Teleport;
});
```

Teleportable

```
pc.script.create('teleportable', function (app) {
  var Teleportable = function (entity) {
    this.entity = entity;
    this.startPosition = null;
    this.lastTeleportFrom = null;
    this.lastTeleportTo = null;
    this.lastTeleport = Date.now();
  };

  Teleportable.prototype = {
    // Called once after all resources are loaded and before the first update
    initialize: function () {
      this.startPosition = this.entity.getPosition().clone();
    },

    update: function() {
      // Make sure we don't fall over. If we do then
      // teleport to the last location
      var pos = this.entity.getPosition();
      if (pos.y < 0) {
        this.teleport(this.lastTeleportFrom, this.lastTeleportTo);
      }
    },

    teleport: function(from, to) {
      // can't teleport too often (500ms)
      if (from && (Date.now() - this.lastTeleport) < 500)
        return;

      // set new teleport time
      this.lastTeleport = Date.now();

      // set last teleport targets
      this.lastTeleportFrom = from;
      this.lastTeleportTo = to;

      // position to teleport to
      var position;

      if (to) {
        // from target
        position = to.getPosition();
        // move a bit higher
        position.y += 0.5;
      } else {
        // to respawn location
        position = this.startPosition;
      }

      // move ball to that point
      this.entity.rigidbody.teleport(position);
      // need to reset angular and linear forces
      this.entity.rigidbody.linearVelocity = pc.Vec3.ZERO;
      this.entity.rigidbody.angularVelocity = pc.Vec3.ZERO;
    }
  };

  return Teleportable;
});
```

movement

```
// Controls the movement speed
pc.script.attribute('speed', 'number', 0.1, {
    min: 0.05,
    max: 0.5,
    step: 0.05,
    decimalPrecision: 2
});

// Script Definition
pc.script.create('movement', function (app) {

    // Creates a new Movement instance
    var Movement = function (entity) {
        this.entity = entity;
        this.force = new pc.Vec3();
    };

    Movement.prototype = {
        // Called every frame, dt is time in seconds since last update
        update: function (dt) {
            var forceX = 0;
            var forceZ = 0;

            // calculate force based on pressed keys
            if (app.keyboard.isPressed(pc.KEY_LEFT)) {
                forceX = -this.speed;
            }

            if (app.keyboard.isPressed(pc.KEY_RIGHT)) {
                forceX += this.speed;
            }

            if (app.keyboard.isPressed(pc.KEY_UP)) {
                forceZ = -this.speed;
            }

            if (app.keyboard.isPressed(pc.KEY_DOWN)) {
                forceZ += this.speed;
            }

            this.force.x = forceX;
            this.force.z = forceZ;

            // if we have some non-zero force
            if (this.force.length()) {

                // calculate force vector
                var rX = Math.cos(-Math.PI * 0.25);
                var rY = Math.sin(-Math.PI * 0.25);
                this.force.set(this.force.x * rX - this.force.z * rY, 0, this.force.z * rX + this.force.x * rY);

                // clamp force to the speed
                if (this.force.length() > this.speed) {
                    this.force.normalize().scale(this.speed);
                }
            }

            // apply impulse to move the entity
            this.entity.rigidbody.applyImpulse(this.force);
        }
    };

    var
```

follow

```
// The entity to follow
pc.script.attribute('target', 'entity', null, {
    displayName: 'Target'
});

// How far from the entity should the follower be
pc.script.attribute('distance', 'number', 4);

// Script Definition
pc.script.create('follow', function (app) {
    // Creates a new Follow instance
    var Follow = function (entity) {
        this.entity = entity;
        this.vec = new pc.Vec3();
    };

Follow.prototype = {
    // Called every frame, dt is time in seconds since last update
    update: function (dt) {
        if (!this.target) return;

        // get the position of the target entity
        var pos = this.target.getPosition();

        // calculate the desired position for this entity
        pos.x += 0.75 * this.distance;
        pos.y += 1.0 * this.distance;
        pos.z += 0.75 * this.distance;

        // smoothly interpolate towards the target position
        this.vec.lerp(this.vec, pos, 0.1);

        // set the position for this entity
        this.entity.setPosition(this.vec);
    }
};

    return Follow;
});
```