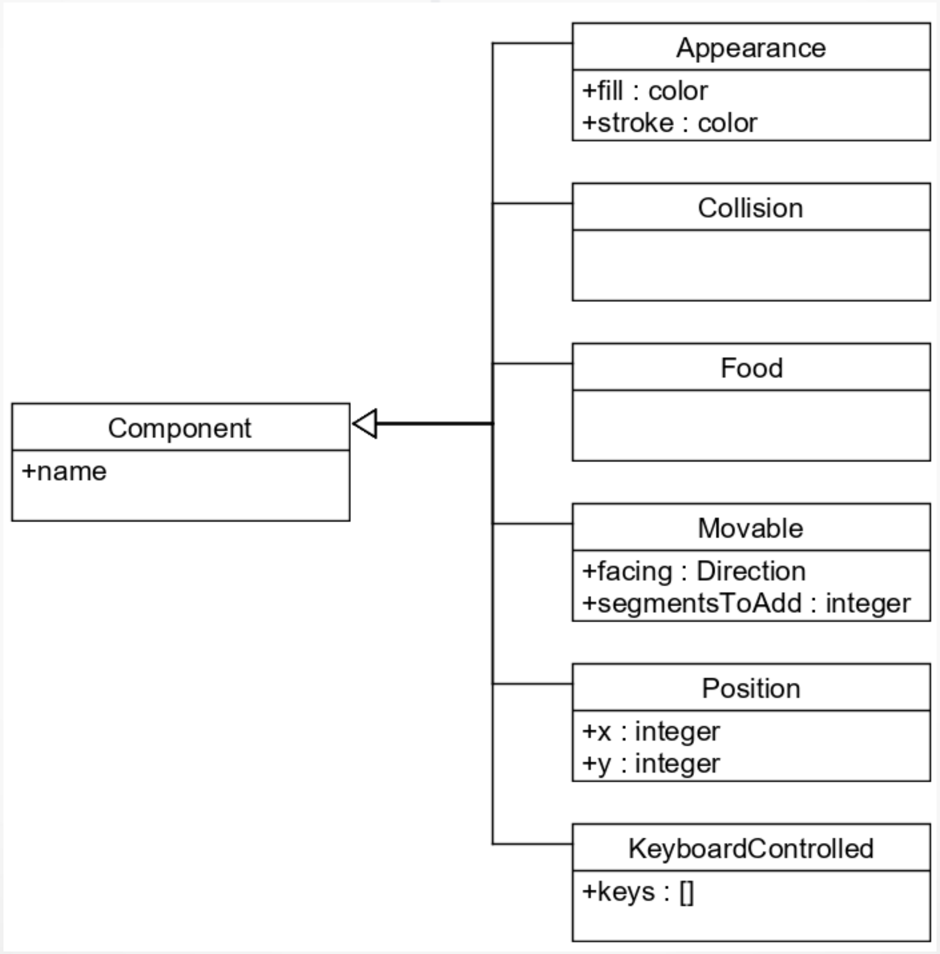
ECS Example : Snake Mini-Game

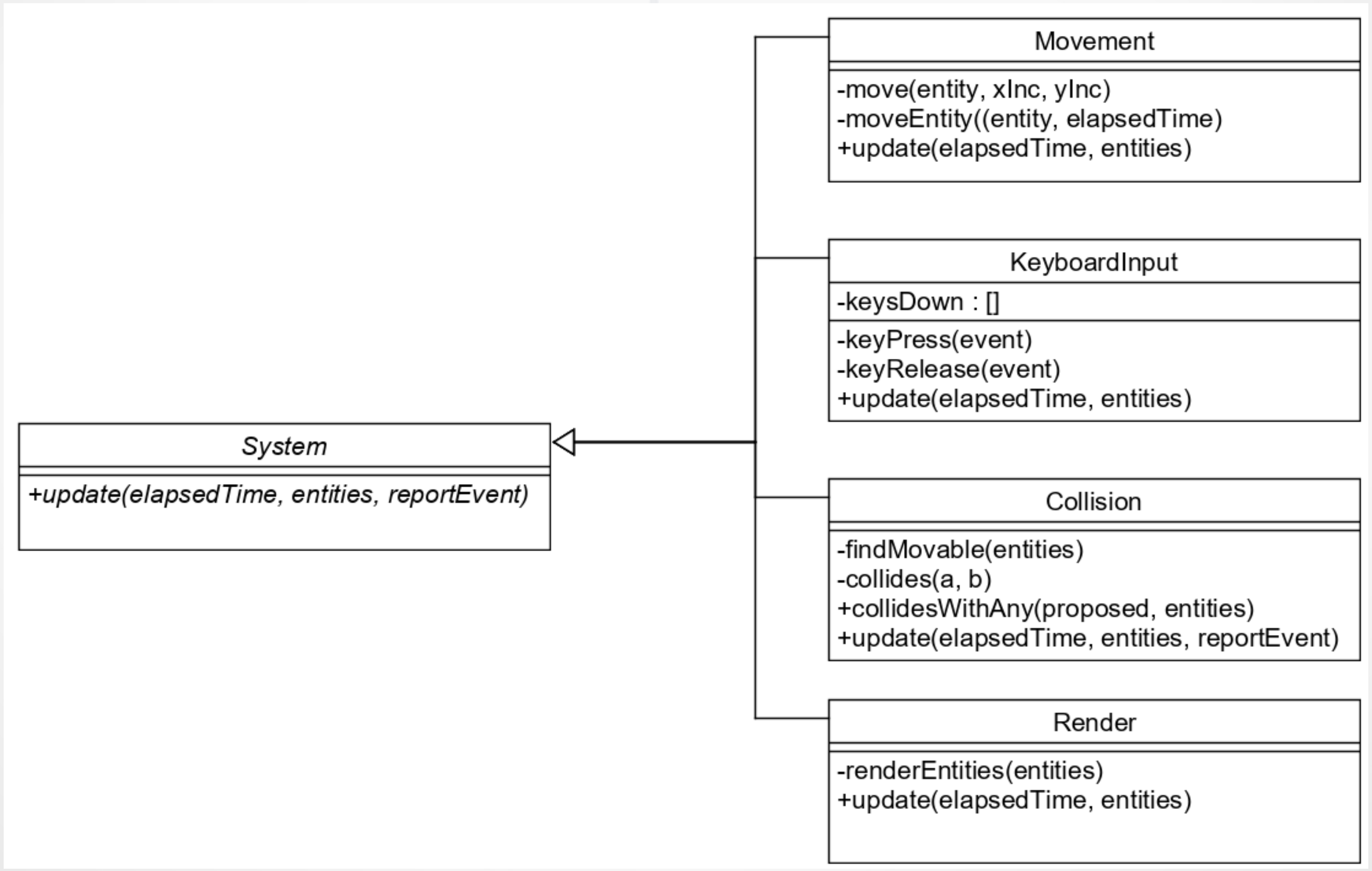
Components


Component
+name
Appearance
+fill : color
+stroke : color
Collision
Food
Movable
+facing : Direction
+segmentsToAdd : integer
Position
+x : integer
+y : integer
KeyboardControlled
+keys : []

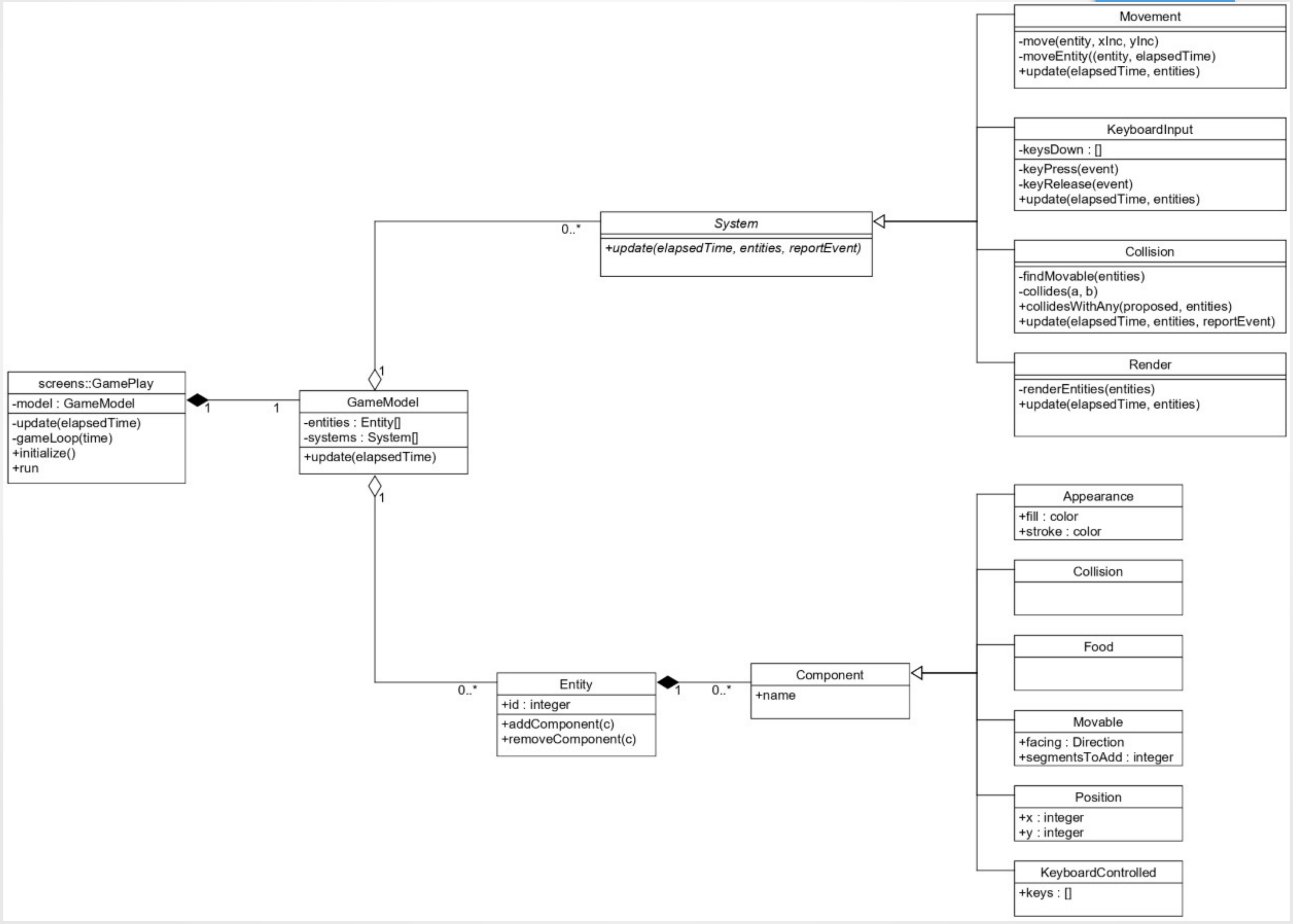
Entities

- **Border** : { Appearance, Position, Collision }
- **Obstacle** : { Appearance, Position, Collision }
- **Food** : { Appearance, Position, Collision, Food }
- **Snake** : { Appearance, Position, Collision, Movable, KeyboardControlled }

Systems


System
+update(elapsedTime, entities, reportEvent)
Movement
-move(entity, xInc, yInc)
-moveEntity((entity, elapsedTime)
+update(elapsedTime, entities)
KeyboardInput
-keysDown : []
-keyPress(event)
-keyRelease(event)
+update(elapsedTime, entities)
Collision
-findMovable(entities)
-collides(a, b)
+collidesWithAny(proposed, entities)
+update(elapsedTime, entities, reportEvent)
Render
-renderEntities(entities)
+update(elapsedTime, entities)

Overview


Movement
-move(entity, xInc, yInc)
-moveEntity((entity, elapsedTime)
+update(elapsedTime, entities)
KeyboardInput
-keysDown : []
-keyPress(event)
-keyRelease(event)
+update(elapsedTime, entities)
System
+update(elapsedTime, entities, reportEvent)
0..*
Collision
-findMovable(entities)
-collides(a, b)
+collidesWithAny(proposed, entities)
+update(elapsedTime, entities, reportEvent)
Render
-renderEntities(entities)
+update(elapsedTime, entities)
screens::GamePlay
-model : GameModel
-update(elapsedTime)
-gameLoop(time)
+initialize()
+run
1
1
GameModel
-entities : Entity[]
-systems : System[]
+update(elapsedTime)
1
1
Appearance
+fill : color
+stroke : color
Collision
Food
Entity
+id : integer
+addComponent(c)
+removeComponent(c)
0..*
1
0..*
Component
+name
Movable
+facing : Direction
+segmentsToAdd : integer
Position
+x : integer
+y : integer
KeyboardControlled
+keys : []

Components

```
MyGame.components.Appearance = function(spec) {
    let api = {
        get name() { return 'appearance'; },
        get fill() { return spec.fill; },
        get stroke() { return spec.stroke; }
    };

    return api;
};
```

```
MyGame.components.Position = function(spec) {
    let api = {
        get name() { return 'position'; },
        get x() { return spec.segments[0].x; },
        get y() { return spec.segments[0].y; },
        get segments() { return spec.segments; }
    };

    return api;
};
```

```
MyGame.components.Collision = function() {
    let api = {
        get name() { return 'collision'; }
    };

    return api;
};
```

Entity Factory

```
let Entity = (function() {
    let nextId = 1;

    function createEntity() {
        let components = {};

        function addComponent(c) {
            components[c.name] = c;
        }

        function removeComponent(c) {
            delete components[c.name];
        }

        return {
            id: nextId++,
            addComponent: addComponent,
            removeComponent: removeComponent,
            get components() { return components; }
        };
    }

    let api = {
        get nextId() { return nextId; },
        createEntity: createEntity
    };

    return api;
}());
```

Creating An Obstacle

```
function createObstacleEntity(x, y) {
    let obstacle = Entity.createEntity();

    obstacle.addComponent(MyGame.components.Appearance({
        fill: {r: 0, g: 255, b: 0 },
        stroke: 'rgb(0, 0, 0)' }));
    obstacle.addComponent(MyGame.components.Position({
        segments: [{ x: x, y: y }] }));
    obstacle.addComponent(MyGame.components.Collision());

    return obstacle;
}
```

Creating A Snake

```
function createSnakeEntity(x, y) {
    let snake = Entity.createEntity();

    snake.addComponent(MyGame.components.Appearance({
        fill: { r: 255, g: 255, b: 255 },
        stroke: 'rgb(0, 0, 0)' }));
    snake.addComponent(MyGame.components.Position({
        segments: [{ x: x, y: y }] }));
    snake.addComponent(MyGame.components.Collision());
    snake.addComponent(MyGame.components.Movable({
        facing: MyGame.constants.Direction.Stopped, moveInterval: MOVE_INTERVAL }));
    let inputSpecification = { keys: {
        'ArrowLeft': MyGame.constants.Direction.Left,
        'ArrowRight': MyGame.constants.Direction.Right,
        'ArrowUp': MyGame.constants.Direction.Up,
        'ArrowDown': MyGame.constants.Direction.Down
    }};
    snake.addComponent(MyGame.components.KeyboardControlled(inputSpecification));

    return snake;
}
```

Implementing a System

...show code in editor...

Update Systems

```
function update(elapsedTime) {
    MyGame.systems.keyboardInput.update(elapsedTime, entities);
    MyGame.systems.movement.update(elapsedTime, entities);
    MyGame.systems.collision.update(elapsedTime, entities, reportEvent);
    MyGame.systems.render.update(elapsedTime, entities);
}
```

Render System

```
function update(elapsedTime) {
    MyGame.systems.keyboardInput.update(elapsedTime, entities);
    MyGame.systems.movement.update(elapsedTime, entities);
    MyGame.systems.collision.update(elapsedTime, entities, reportEvent);
    MyGame.systems.render.update(elapsedTime, entities);
}
```