

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe Car encapsula la lògica que controla un vehicle dins d'un joc de curses, amb un sistema dinàmic que gestiona el moviment, la degradació de components i les interaccions entre aquests. La classe es basa en altres classes de components (com Engine, Tires, Brakes, etc.) i utilitza efectes especials per modificar el comportament del vehicle en funció de l'estat dels components.

Localització: io/github/revNrun/revNrun/model/car/Car.java class Car

Test: io/github/revNrun/revNrun/model/car/CarTest.java. En aquesta classe s'ha realitzat TDD i test de caixa negra amb particions equivalents, valors límit i frontera, pairwise testing i statement coverage

Element ^	Class, %	Method, %	Line, %	Branch, %
Car	100% (2/2)	95% (60/63)	97% (194/200)	97% (99/102)

Statement coverage del 97% en la classe Car.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares


Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: És la base per als components dels vehicles, com motors, rodes o altres peces que poden tenir durabilitat, desgastar-se amb l'ús, i necessitar reparacions. Les subclasses de `AbstractComponent` implementen comportaments específics per a diferents tipus de components, però comparteixen la funcionalitat comuna de gestió de durabilitat i desgast.

Localització: `io/github/revNrun/revNrun/model/car/components/AbstractComponent.java`

Test: `io/github/revNrun/revNrun/model/car/components/AbstractComponentTest.java`. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra, però el tipus de test realitzat és principalment caixa blanca, ja que es prova el comportament intern de la classe `AbstractComponent` mitjançant l'execució de mètodes específics i la verificació de l'estat intern de l'objecte després d'executar-los (per exemple, comprovant si els valors de durabilitat es comporten correctament segons les condicions). S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, mockups amb tres mock objects i statement coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
 AbstractComponent	100% (1/1)	90% (9/10)	95% (21/22)	100% (14/14)

Statement coverage del 95% en la classe `AbstractComponent`.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe WheelMountedComponent és una classe base per a components muntats a les rodes d'un vehicle, amb funcionalitats per controlar el desgast i obtenir informació sobre l'eix i el costat del component.

Localització:

io/github/revNrun/revNrun/model/car/components/WheelMountedComponent.java

Test:

io/github/revNrun/revNrun/model/car/components/WheelMountedComponentTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat tant test de caixa negra com blanca, utilitzant les tècniques de particions equivalents, valors límit i frontera, mockups i statement coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
🔗 WheelMountedComponent	100% (1/1)	100% (4/4)	100% (8/8)	100% (4/4)

Statement coverage del 100% en la classe WheelMountedComponent.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe Effect emmagatzema un efecte específic (identificat per un tipus d'efecte) i un valor associat a aquest efecte. El valor de l'efecte està restringit dins d'un rang definit per les constants MAX_VALUE i MIN_VALUE. Quan es crea un objecte de tipus Effect, es pot passar un valor per establir aquest efecte, però aquest valor s'ajusta automàticament per assegurar-se que es mantingui dins dels límits mínim i màxim. A més, la classe permet obtenir i modificar el valor de l'efecte.

Localització: io/github/revNrun/revNrun/model/car/components/Effect.java

Test: io/github/revNrun/revNrun/model/car/components/EffectTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra: els tests es centren en l'entrada (els paràmetres passats al constructor) i l'output (els valors que es retornen a través dels mètodes getEffect i getValue). S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, statement coverage i decision coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
© Effect	100% (1/1)	75% (3/4)	87% (7/8)	100% (2/2)

Statement coverage del 87% en la classe Effect.

```
5 public class Effect {
6     private static final float MAX_VALUE = 20f; 1 usage
7     private static final float MIN_VALUE = -20f; 2 usages
8
9     private final EffectType effect; 2 usages
10    private float value; 4 usages
11
12    public Effect(EffectType effect, float value) {
13        this.effect = effect;
14        if (value < MIN_VALUE) {
15            this.value = MIN_VALUE;
16        } else this.value = Math.min(value, MAX_VALUE);
17    }
18
19    public EffectType getEffect() { return effect; }
22
23    public float getValue() { return value; }
26
27    public void setValue(float value) { this.value = value; }
30 }
```

Condition coverage en la classe Effect.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe Checkpoints gestiona els punts de control d'un circuit en un joc. La classe proporciona funcionalitats per verificar si un punt es troba dins del circuit, seguir el progrés dels punts de control, determinar l'estat de la volta, i gestionar els punts de control saltats.

Localització: io/github/revNrun/revNrun/model/checkpoints/Checkpoints.java

Test: io/github/revNrun/revNrun/model/checkpoints/CheckpointsTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat sobretot test de caixa negra que verifiquen si els resultats obtinguts són correctes per a diferents entrades, com es veu en tests com lapStatus(), isInsideCircuit(), i la gestió de les excepcions. S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, loop testing, statement coverage i path coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
© Checkpoints	100% (1/1)	100% (10/10)	98% (59/60)	86% (40/46)

Statement coverage del 98% en la classe Checkpoints.

```
60 public boolean isInsideCircuit(Vector2 point) { 108 usages
61     if (point == null) {
62         throw new IllegalArgumentException("Point must be not null.");
63     }
64
65     Vector2 closestCheckPoint = null;
66     float minDistance = Float.MAX_VALUE;
67
68     for (Vector2 p : checkPoints) {
69         float distance = p.distance(point);
70         if (distance <= width && distance < minDistance) {
71             minDistance = distance;
72             closestCheckPoint = p;
73         }
74     }
75
76     if (closestCheckPoint != null) {
77         recordProgress(closestCheckPoint);
78         return true;
79     }
80
81     return false;
82 }
```

Path coverage i loop testing en el mètode isInsideCircuit de la classe Checkpoints.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

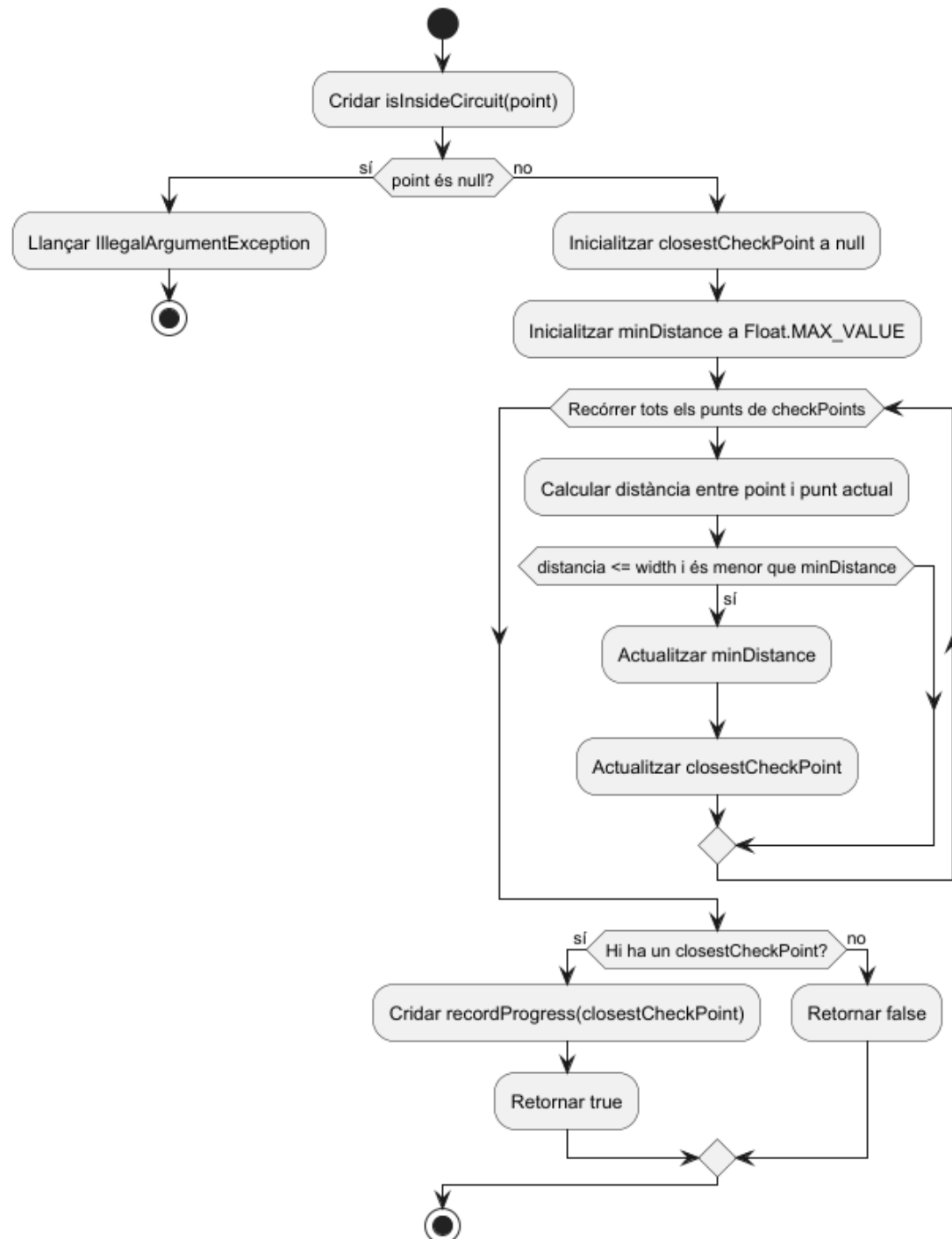


Diagrama d'activitats del path coverage del mètode `isInsideCircuit` de la classe `Checkpoints`.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe GhostCar té com a objectiu registrar i reproduir el comportament d'un cotxe "fantasma" en una cursa.

Localització: io/github/revNrun/revNrun/model/ghost_car/GhostCar.java

Test: io/github/revNrun/revNrun/model/ghost_car/GhostCarTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra, i també de caixa blanca, com la validació de la consistència en l'índex d'estat o l'ús de les funcions nextFrame i reset. S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, mockups, statement coverage i decision coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
© GhostCar	100% (1/1)	100% (15/15)	100% (39/39)	100% (14/14)

Statement coverage del 100% en la classe Checkpoints.

```
30      public void nextFrame() { 9 usages
31          if (notEmpty()) {
32              currentStateIndex++;
33              int size = states.size();
34              if (currentStateIndex >= size) {
35                  currentStateIndex = size - 1;
36              }
37          }
38      }
```

Decision coverage en el mètode nextFrame() de la classe GhostCar.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe GhostState representa l'estat d'un "ghost car" (cotxe fantasma) en un moment específic dins el joc. La funcionalitat principal de la classe és encapsular la posició, l'angle de rotació i el moment temporal d'aquest estat, permetent que es pugui emmagatzemar i recuperar per representar el moviment o traça d'un cotxe fantasma.

Localització: io/github/revNrun/revNrun/model/ghost_car/GhostState.java

Test: io/github/revNrun/revNrun/model/ghost_car/GhostStateTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa blanca, i s'han utilitzat les tècniques de particions equivalents, valors límit i frontera, i statement coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
© GhostState	100% (1/1)	100% (7/7)	88% (15/17)	50% (2/4)

Statement coverage del 88% en la classe GhostState.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe LapTimer actua com un cronòmetre per mesurar i gestionar el temps de volta en una cursa.

Localització: io/github/revNrun/revNrun/model/lap_timer/LapTimer.java

Test: io/github/revNrun/revNrun/model/lap_timer/LapTimerTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra i blanca, i s'han utilitzat les tècniques de particions equivalents, valors límit i frontera, statement coverage i decision coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
© LapTimer	100% (1/1)	100% (11/11)	100% (28/28)	90% (9/10)

Statement coverage del 100% en la classe LapTimer.

```
29      public String getCurrentLapTime() { 12 usages
30          if (!isRunning) {
31              if (lastLapTime == 0) {
32                  return "00:00.000";
33              }
34              return formatTime(lastLapTime);
35          }
36          long currentLapTime = System.nanoTime() - startTime;
37          return formatTime(currentLapTime);
38      }
```

Decision coverage en el mètode getCurrentLapTime() de la classe LapTime.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe SimplexNoise implementa el soroll Simplex en 2D, una tècnica matemàtica que genera patrons de soroll suaus i naturals.

Localització: [io/github/revNrun/revNrun/model/track/SimplexNoise.java](https://github.com/revNrun/revNrun/blob/master/model/track/SimplexNoise.java)

Test: [io/github/revNrun/revNrun/model/track/SimplexNoiseTest.java](https://github.com/revNrun/revNrun/blob/master/model/track/SimplexNoiseTest.java). En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra, i s'han utilitzat les tècniques de particions equivalents, valors límit i frontera i statement coverage.

Element ^	Class, %	Method, %	Line, %	Branch, %
© SimplexNoise	100% (1/1)	100% (4/4)	100% (43/43)	100% (12/12)

Statement coverage del 100% en la classe SimplexNoise.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe `RandomTrackPoints` genera punts aleatoris per al circuit. Els punts generats defineixen la forma general del circuit, que inclou corbes suaus i trajectes tancats. La classe segueix un procés estructurat per generar diferents tipus de punts i utilitza algoritmes com el *Simplex Noise* i interpolació suau (*Catmull-Rom*) per assegurar que el circuit sigui fluid i natural.

Localització: `io/github/revNrun/revNrun/model/track/RandomTrackPoints.java`

Test: `io/github/revNrun/revNrun/model/track/RandomTrackPointsTest.java`. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra i de caixa blanca, com els de comprovació d'interval, absència d'interseccions, i tancament del circuit. S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, statement coverage i decision coverage.

Element ^	Class, %	Method,...	Line, %	Branch, %
© RandomTrackPoints	100% (1/1)	100% (17/...	97% (69/71)	80% (16/20)

Statement coverage del 97% en la classe `RandomTrackPoints`.

```
240     private void adjustBasePoints() { 1 usage
241         // Check the base control points to not have a near point too close. If so, remove it from the list.
242         basePoints = AdjustPoints.adjustNearPoints(basePoints, controlPointMinDistance);
243
244         // Check the base control points to not intersect. If so, remove segments that create an intersection.
245         basePoints = AdjustPoints.adjustIntersections(basePoints, controlPointMinDistance);
246
247         if (basePoints.size() <= 3) {
248             RandomTrackPoints newTrack = new RandomTrackPoints();
249             basePoints = newTrack.getBasePoints();
250         }
251     }
```

Decision coverage en el mètode `adjustBasePoints()` de la classe `RandomTrackPoints`.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe TrackSmoothing s'encarrega de suavitzar un conjunt de punts de control mitjançant l'algorisme de Catmull-Rom. Aquesta tècnica genera corbes suaus entre punts de control per a crear una pista contínua sense angles brusc.

Localització: io/github/revNrun/revNrun/model/track/TrackSmoothing.java

Test: io/github/revNrun/revNrun/model/track/TrackSmoothingTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat majoritàriament test de caixa negra, però també de caixa blanca, com en el mètode testOuterLoopExecution. S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, loop testing, statement coverage.

Element ^	Class, %	Method...	Line, %	Branch, %
© TrackSmoothing	100% (1/1)	100% (4/4)	100% (39/...	100% (14/14)

Statement coverage del 100% en la classe TrackSmoothing.

```
44 for (int i = 1; i < numSegments - 1; i++) {
45     // We use the first and last points to set the start and end direction of the curve that goes from p1 to p2
46     p0 = controlPoints.get(i - 1);
47     p1 = controlPoints.get(i);
48     p2 = controlPoints.get(i + 1);
49     p3 = controlPoints.get(i + 2);
50     // Add the control point of the start of the curve of every segment
51     // (the end of the curve of the past segment)
52     if (addControlPoints) smoothedPoints.add(p1);
53     // Calculate the distance between the control points
54     distance = p1.distance(p2);
55     // If the distance is less or equal than interpolatedDistance, no need of interpolation,
56     // we can skip the rest of the iteration
57     if (distance <= interpolatedDistance) continue;
58     // Calculate the number of necessary segments
59     numSegmentsNeeded = (int) Math.ceil(distance / interpolatedDistance);
60     numSamples = numSegmentsNeeded - 1;
61     // Curve factor to adjust the number of points on heavy curves
62     curveRatio = calculateCurveRatio(p0, p1, p2, p3);
63     numSamples = Math.round(numSamples * curveRatio);
64     for (int j = 1; j <= numSamples; j++) {
65         float t = (float) j / (float) (numSamples + 1);
66         Vector2 point = computeCatmullRomPoint(p0, p1, p2, p3, t);
67         smoothedPoints.add(point);
68     }
69 }
```

Loop testing en el bucle aniuat del mètode computeCatmullRom() de la classe trackSmoothing.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe Track representa el circuit de carreres. Té com a funció principal gestionar els punts de control del circuit, calcular-ne el radi màxim i generar-ne els límits esquerre i dret.

Localització: io/github/revNrun/revNrun/model/track/Track.java

Test: io/github/revNrun/revNrun/model/track/TrackTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat tant test de caixa negra, com en testTrackInitialization o testTrackBordersHaveSameSizeAsTrackPoints, com de caixa blanca, com en els mètodes testOuterLoopZeroIterations o testInnerLoopMultipleIterations. S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, loop testing, statement coverage.

Element ^	Class, %	Method...	Line, %	Branch, %
© Track	100% (1/1)	100% (9/9)	93% (27/29)	72% (13/18)

Statement coverage del 93% en la classe Track.

```
60      for (int i = 0; i < controlPoints.size(); i++) {
61          for (int j = i + 1; j < controlPoints.size(); j++) {
62              float distance = controlPoints.get(i).distance(controlPoints.get(j));
63              if (distance > maxDistance) {
64                  maxDistance = distance;
65              }
66          }
67      }
```

Loop testing aniuat en el mètode calculateRadius() de la classe Track.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe BorderGenerator genera les vores esquerra i dreta del circuit 2D, basant-se en una llista de punts que defineixen el traçat principal del circuit.

Localització: io/github/revNrun/revNrun/model/track/BorderGenerator.java

Test: io/github/revNrun/revNrun/model/track/BorderGeneratorTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat majoritàriament test de caixa negra, i també de caixa blanca, com en el mètode testAllPointsTangentAndOffset. S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, loop testing, statement coverage.

Element ^	Class, %	Metho...	Line, %	Branch, %
© BorderGenerator	100% (1/1)	100% (4/4)	95% (21/2...	94% (17/18)

Statement coverage del 95% en la classe BorderGenerator.

```
26 | for (int i = 0; i < points.size(); i++) {
27 |     Vector2 current = points.get(i);
28 |     Vector2 previous = i > 0 ? points.get(i - 1) : null;
29 |     Vector2 next = i < points.size() - 1 ? points.get(i + 1) : null;
30 |
31 |     Vector2 tangent = getTangent(previous, current, next);
32 |     Vector2 normal = new Vector2(-tangent.getY(), tangent.getX()).nor();
33 |     Vector2 borderPoint = current.cpy().add(normal.scl(offset));
34 |
35 |     borderPoints.add(borderPoint);
36 | }
```

Loop testing en el mètode generateBorder() de la classe BorderGenerator.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe Vector2 representa un vector en un espai bidimensional amb components x i y . Aquesta classe proporciona diverses operacions bàsiques sobre vectors, com ara el càlcul de la distància, l'angle entre vectors, la normalització, la suma, la resta, l'escalat i altres mètodes geomètrics.

Localització: `io/github/revNrun/revNrun/model/vector/Vector2.java`

Test: `io/github/revNrun/revNrun/model/vector/Vector2Test.java`. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra, per exemple, es comprova el comportament de la funció `distance()` o `angleBetween()` amb diferents vectors d'entrada sense observar la seva implementació interna. S'han utilitzat les tècniques de particions equivalents, valors límit i frontera, i statement coverage.

Element ^	Class, %	Metho...	Line, %	Branch,...
© Vector2	100% (1/1)	92% (24...	86% (80/...	61% (70/...

Statement coverage del 86% en la classe Vector2.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe CreateCar té la responsabilitat de crear el vehicle amb diferents components (com ara el motor, el xassís, les rodes, la suspensió, els frens, etc.), a més de gestionar els efectes associats a cada component.

Localització: io/github/revNrun/revNrun/model/CreateCar.java

Test: io/github/revNrun/revNrun/model/CreateCarTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra. S'han utilitzat les tècniques de particions equivalents i statement coverage.

Element ^	Class, %	Metho...	Line, %	Branch,...
🕒 CreateCar	100% (1...	100% (4...	100% (5...	100% (0/0)

Statement coverage del 100% en la classe CreateCar.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe CountdownController és el controlador que gestiona el compte enrere del joc, implementant una animació per mostrar els números en compte enrere abans de començar la partida.

Localització: io/github/revNrun/revNrun/controllers/game/CountdownController.java

Test: io/github/revNrun/revNrun/controllers/game/CountdownControllerTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra. S'han utilitzat les tècniques de particions equivalents, mockups amb mockito, valors límit i frontera, statement coverage i condition coverage.

Element ^	Class,...	Meth...	Line, %	Branch...
© CountdownController	100% (1...	100% (...)	100% (4...	92% (13...

Statement coverage del 100% en la classe CountdownController.

```
55 private void handleNumberTransition(int number, float cycleTime) { 3 usages
56     currentNumber = number;
57
58     if (cycleTime < zoomTime) {
59         float progress = cycleTime / zoomTime;
60         float scale = Interpolation.swingOut.apply(start: 0.5f, end: 1.5f, progress);
61         countdownView.setScale(scale);
62         countdownView.setNumberAlpha(1f);
63     } else {
64         float fadeProgress = (cycleTime - zoomTime) / fadeTime;
65         countdownView.setNumberAlpha(1 - fadeProgress);
66         countdownView.setScale(1.5f);
67     }
68 }
```

Condition coverage en el mètode handleNumberTransition() de la classe Countdowncontroller.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe TrackController és la responsable de controlar la interacció entre el cotxe i la pista del joc.

Localització: io/github/revNrun/revNrun/controllers/game/track/TrackController.java

Test: io/github/revNrun/revNrun/controllers/game/track/TrackControllerTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra i de caixa blanca. S'han utilitzat les tècniques de particions equivalents, mockups amb mockito, valors límit i frontera, i statement coverage.

Element ^	Class,...	Meth...	Line, %	Branch...
© TrackController	100% (1...	100% (1...	90% (30...	62% (5/8)

Statement coverage del 90% en la classe TrackController.

Leonardo Ruben Edenak Chouev

Ferran Sogas Linares

Dijous 12:30 – 14:30

Rev & Run

Funcionalitat: La classe CarController és la responsable de gestionar el comportament del cotxe.

Localització: io/github/revNrun/revNrun/controllers/game/car/CarController.java

Test: io/github/revNrun/revNrun/controllers/game/car/CarControllerTest.java. En aquesta classe s'ha realitzat TDD. S'ha realitzat test de caixa negra i de caixa blanca, com a currentDurability. S'han utilitzat les tècniques de particions equivalents, mockups amb mockito, valors límit i frontera, statement coverage i decision coverage.

Element ^	Class,...	Meth...	Line, %	Branc...
© CarController	100% (...)	100% (...)	100% (8...	96% (2...

Statement coverage del 100% en la classe CarController.

```
89 private void updatePosition(float delta) { 1 usage
90     Vector2 pos = new Vector2(car.getPosition());
91     if (car.getPositionX() < 0) {
92         pos.setX(0);
93     } else if (car.getPositionX() > ViewUtils.WORLD_WIDTH - carView.getCarWidth()) {
94         pos.setX(ViewUtils.WORLD_WIDTH - carView.getCarWidth());
95     }
96
97     if (car.getPositionY() < 0) {
98         pos.setY(0);
99     } else if (car.getPositionY() > ViewUtils.WORLD_HEIGHT - carView.getCarHeight()) {
100         pos.setY(ViewUtils.WORLD_HEIGHT - carView.getCarHeight());
101     }
102     car.setPosition(pos);
103     car.updatePosition(delta);
104 }
```

Decision coverage al mètode updatePosition() de la classe CarController.