




 BuzzyBeetleCollider
<ul style="list-style-type: none"> ◇ buzzyBeetle: BuzzyBeetle
<ul style="list-style-type: none"> ● BuzzyBeetleCollider(z: BuzzyBeetle, b: Rectangle) ● getEntity(): BuzzyBeetle ● getCollision(): BuzzyBeetleCollision ● recieveCollision(c: VisitorCollision, a: Axis)


 EnemyCollider
<ul style="list-style-type: none"> ◇ <u>DISPLACEMENT_COEFFICIENT: int</u>
<ul style="list-style-type: none"> ● Enemy getEntity(): abstract ● EnemyCollider(b: Rectangle) ◇ kill(mario: Mario) ◇ getKilled(mario: Mario, sound: String) ● calculateCollisionDirection(m: MarioCollision): Direction ● handleHorizontalCollision(m: MarioCollision) ● handleVerticalCollision(m: MarioCollision) ● handleHorizontalCollision(m: SuperMarioCollision) ● handleVerticalCollision(m: SuperMarioCollision) ● handleVerticalCollision(m: InvulnerableCollision) ● handleHorizontalCollision(m: InvulnerableCollision) ● handleHorizontalCollision(m: StarMarioCollision) ● handleVerticalCollision(m: StarMarioCollision) ● handleHorizontalCollision(f: FireBallCollision) ● handleVerticalCollision(f: FireBallCollision) ● handleVerticalCollision(e: EnemyCollision) ● handleVerticalCollision(s: ShellEnemyCollision) ◇ bounce(e: EnemyCollider) ● handleHorizontalCollision(s: ShellEnemyCollision) ● handleHorizontalCollision(e: EnemyCollision)


 GoombaCollider
<ul style="list-style-type: none"> ◇ goomba: Goomba
<ul style="list-style-type: none"> ● GoombaCollider(g: Goomba, b: Rectangle) ● getEntity(): Goomba ● getCollision(): GoombaCollision ● recieveCollision(c: VisitorCollision, a: Axis)


 KoopaTroopaCollider
<ul style="list-style-type: none"> ◇ koopa: KoopaTroopa
<ul style="list-style-type: none"> ● KoopaTroopaCollider(k: KoopaTroopa, b: Rectangle) ● getEntity(): KoopaTroopa ● getCollision(): KoopaTroopaCollision ● recieveCollision(c: VisitorCollision, a: Axis)


 LakituCollider
<ul style="list-style-type: none"> ◇ lakitu: Lakitu
<ul style="list-style-type: none"> ● LakituCollider(l: Lakitu, b: Rectangle) ● getEntity(): Lakitu ● getCollision(): LakituCollision ● recieveCollision(c: VisitorCollision, a: Axis)


 PiranhaPlantCollider
<ul style="list-style-type: none"> ◇ piranha: PiranhaPlant
<ul style="list-style-type: none"> ● PiranhaPlantCollider(p: PiranhaPlant, b: Rectangle) ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): PiranhaPlantCollision ● getEntity(): PiranhaPlant ● handleVerticalCollision(m: InvulnerableCollision) ● handleVerticalCollision(m: MarioCollision) ● handleVerticalCollision(m: SuperMarioCollision)


 ShellEnemyCollider
<ul style="list-style-type: none"> ● ShellEnemy getEntity(): abstract ● ShellEnemyCollider(b: Rectangle) ● handleHorizontalCollision(m: MarioCollision) ● handleHorizontalCollision(m: SuperMarioCollision) ● handleHorizontalCollision(m: StarMarioCollision) ● handleVerticalCollision(m: StarMarioCollision) ● handleHorizontalCollision(f: FireBallCollision) ● handleVerticalCollision(f: FireBallCollision)


 SpinyCollider
<ul style="list-style-type: none"> ◇ spiny: Spiny
<ul style="list-style-type: none"> ● SpinyCollider(s: Spiny, b: Rectangle) ● getEntity(): Spiny ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): SpinyCollision ● handleVerticalCollision(m: MarioCollision)


 DeleterCollider
<ul style="list-style-type: none"> DeleterCollider(b: Rectangle) getEntity(): Entity recieveCollision(c: VisitorCollision, a: Axis) getCollision(): DeleterCollision delete(c: VisitorCollision) handleHorizontalCollision(c: UpdateableEntityCollision) handleHorizontalCollision(c: VisitorCollision)


 EmptyBlockCollider
<ul style="list-style-type: none"> block: EmptyBlock
<ul style="list-style-type: none"> EmptyBlockCollider(e: EmptyBlock, b: Rectangle) getEntity(): EmptyBlock recieveCollision(c: VisitorCollision, a: Axis) getCollision(): EmptyBlockCollision handleVerticalCollision(m: MarioCollision) handleHorizontalCollision(m: MarioCollision) handleHorizontalCollision(p: UpdateableEntityCollision) handleVerticalCollision(p: UpdateableEntityCollision) handleHorizontalCollision(f: FireBallCollision) handleVerticalCollision(f: FireBallCollision)

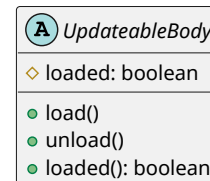
 GraphicUnloaderCollider
<ul style="list-style-type: none"> GraphicUnloaderCollider(b: Rectangle) getEntity(): Entity recieveCollision(c: VisitorCollision, a: Axis) getCollision(): UnloaderCollision handleHorizontalCollision(c: VisitorCollision)

 LevelEndCollider
<ul style="list-style-type: none"> LevelEndCollider(b: Rectangle) getEntity(): Entity recieveCollision(c: VisitorCollision, a: Axis) getCollision(): LevelEndCollision handleHorizontalCollision(c: ScreenDisplacementCollision)

 LoaderCollider
<ul style="list-style-type: none"> LoaderCollider(b: Rectangle) getEntity(): Entity recieveCollision(c: VisitorCollision, a: Axis) getCollision(): LoaderCollision handleHorizontalCollision(c: UpdateableEntityCollision)

 ScreenBorderCollider
<ul style="list-style-type: none"> position: Direction
<ul style="list-style-type: none"> ScreenBorderCollider(b: Rectangle, position: Direction) getEntity(): Entity recieveCollision(c: VisitorCollision, a: Axis) getCollision(): ScreenBorderCollision handleHorizontalCollision(m: MarioCollision) handleHorizontalCollision(c: VisitorCollision) horizontalCollision(c: VisitorCollision) handleHorizontalCollision(p: PiranhaPlantCollision) handleVerticalCollision(p: PiranhaPlantCollision) handleHorizontalCollision(c: LevelEndCollision) handleVerticalCollision(c: LevelEndCollision) handleHorizontalCollision(c: EmptyBlockCollision) handleVerticalCollision(c: EmptyBlockCollision)

 ScreenDisplacementCollider
<ul style="list-style-type: none"> leftBorder: ScreenBorderCollider rightBorder: ScreenBorderCollider loader: LoaderCollider unloader: GraphicUnloaderCollider deleter: DeleterCollider
<ul style="list-style-type: none"> ScreenDisplacementCollider(getEntity(): Entity getCollision(): ScreenDisplacementCollision recieveCollision(c: VisitorCollision, a: Axis) handleHorizontalCollision(m: MarioCollision)



C Block

- ◇ SPRITES_FOLDER: String
- ◇ collider: BlockCollider
- ◇ graphicElement: GameGraphicElement

- Block()
- getGraphicElement(): GameGraphicElement
- getCollider(): BlockCollider

C Brick

- ◇ SPRITES_FOLDER: String
- String: final
- ◇ collider: BrickCollider
- ◇ graphicElement: GameGraphicElement

- Brick()
- getGraphicElement(): GameGraphicElement
- getCollider(): BrickCollider

C Castle

- ◇ String: final
- ◇ sprite: String

- Castle(c: char)

C Pipe

- ◇ FOLDER_PATH: String
- ◇ TOP_PIPE: String
- ◇ BASE_PIPE: String
- ◇ collider: PipeCollider
- ◇ graphicElement: GameGraphicElement

- Pipe(c: char)
- getGraphicElement(): GameGraphicElement
- getCollider(): PipeCollider

C QuestionBlock

- ◇ SPRITES_FOLDER: String
- int: final
- ◇ collider: QuestionBlockCollider
- ◇ graphicElement: GameGraphicElement
- ◇ depends: boolean
- ◇ active: boolean
- ◇ entity: ObserverUpdateableEntity
- ◇ animator: MovementAnimator

- final List<String> ANIMATED_SPRITES = List.of(
- QuestionBlock(s: char)
- getGraphicElement(): GameGraphicElement
- getCollider(): QuestionBlockCollider
- setDepends(b: boolean)
- getDepends(): boolean
- interaction(p: PowerUp)
- getActive(): boolean
- update()

A

BaseCollider

◇ bounds: Rectangle

◇ activated: boolean

◇ velocity: Vector2D

◇ colliding: boolean

◇ moving: boolean

◇ nextVelocity: Vector2D

● BaseCollider(b: Rectangle)

● isMoving(): boolean

● setMoving(m: boolean)

● isColliding(): boolean

● setColliding(c: boolean)

● getPosition(): Point

● setPosition(x: int, y: int)

● getVelocity(): Vector2D

● getNextVelocity(): Vector2D

● isActivated(): boolean

● setActivated(status: boolean)

● activate()

● deactivate()

● getBounds(): Rectangle

● translate(dx: int, dy: int)

● moveX()

● moveY()

● updateVelocity()

● getSize(): Dimension

● setSize(width: int, height: int)

● displaceX(intersection: Rectangle, coefficient: int): int

● displaceY(intersection: Rectangle, coefficient: int): int

● copy(c: Collider)

● track(c: Collider)

● handleHorizontalCollision(c: VisitorCollision)

● handleVerticalCollision(c: VisitorCollision)

C

CoinCollider

◇ coin: Coin

● CoinCollider(c: Coin, b: Rectangle)

● getEntity(): Coin

● recieveCollision(c: VisitorCollision, a: Axis)

● getCollision(): CoinCollision

● handleHorizontalCollision(m: MarioCollision)

● handleVerticalCollision(m: MarioCollision)

I

Collider

● getEntity(): Entity

● recieveCollision(c: VisitorCollision, a: Axis)

● getBounds(): Rectangle

● getCollision(): VisitorCollision

● setPosition(x: int, y: int)

● getPosition(): Point

● translate(dx: int, dy: int)

● getSize(): Dimension

● setSize(width: int, height: int)

● getVelocity(): Vector2D

● getNextVelocity(): Vector2D

● isActivated(): boolean

● setActivated(status: boolean)

● activate()

● deactivate()

● updateVelocity()

● moveX()

● moveY()

● setColliding(c: boolean)

● isColliding(): boolean

● isMoving(): boolean

● setMoving(m: boolean)

● displaceX(intersection: Rectangle, coefficient: int): int

● displaceY(intersection: Rectangle, coefficient: int): int

● copy(c: Collider)

● track(c: Collider)

C

FireBallCollider

◇ fireBall: FireBall

● FireBallCollider(f: FireBall, b: Rectangle)

● recieveCollision(c: VisitorCollision, a: Axis)

● getCollision(): FireBallCollision

● getEntity(): FireBall

C

FlagPoleCollider

◇ flagPole: FlagPole

● FlagPoleCollider(e: FlagPole, b: Rectangle)

● getEntity(): Entity

● recieveCollision(c: VisitorCollision, a: Axis)

● getCollision(): VisitorCollision

● handleHorizontalCollision(m: MarioCollision)

C

LanguageSwitcherCollider

◇ block: ConfigurationBlock

● LanguageSwitcherCollider(e: ConfigurationBlock, b: Rectangle)

● getEntity(): ConfigurationBlock

● recieveCollision(c: VisitorCollision, a: Axis)

● getCollision(): LanguageSwitcherCollision

● handleVerticalCollision(m: MarioCollision)

C

ModeSwitcherCollider

◇ block: ConfigurationBlock

● ModeSwitcherCollider(q: ConfigurationBlock, b: Rectangle)

● getEntity(): ConfigurationBlock

● recieveCollision(c: VisitorCollision, a: Axis)

● getCollision(): VisitorCollision

● handleVerticalCollision(m: MarioCollision)

I

MovableCollider

C

RankingShowCollider

◇ block: ConfigurationBlock

● RankingShowCollider(q: ConfigurationBlock, b: Rectangle)

● getEntity(): ConfigurationBlock

● recieveCollision(c: VisitorCollision, a: Axis)

● getCollision(): VisitorCollision

● handleVerticalCollision(m: MarioCollision)

C FireFlower
<ul style="list-style-type: none"> ◇ SPRITES_FOLDER: String ○ final FRAMES_PER_SPRITE: int ○ final POINTS_MARIO: int ○ final POINTS_SUPER_MARIO: int ○ final POINTS_FIRE_MARIO: int ◇ collider: FireFlowerCollider ◇ graphicElement: GameGraphicElement ◇ animator: MovementAnimator
<ul style="list-style-type: none"> ● final List<String> ANIMATED_SPRITES = List.of(FireFlower()) ● update() ● getGraphicElement(): GameGraphicElement ● getCollider(): FireFlowerCollider

C GreenMushroom
<ul style="list-style-type: none"> ◇ SPRITES_FOLDER: String ○ final POINTS: int ◇ collider: GreenMushroomCollider ◇ graphicElement: GameGraphicElement
<ul style="list-style-type: none"> ● GreenMushroom() ● getGraphicElement(): GameGraphicElement ● getCollider(): GreenMushroomCollider

I PowerUp
<ul style="list-style-type: none"> ○ String: final

C Star
<ul style="list-style-type: none"> ○ String: final ○ int: final ○ final POINTS_MARIO: int ○ final POINTS_SUPER_MARIO: int ○ final POINTS_STAR_MARIO: int ○ final BOUNCE_SPEED: int ◇ collider: StarCollider ◇ graphicElement: GameGraphicElement ◇ animator: MovementAnimator
<ul style="list-style-type: none"> ● final List<String> ANIMATED_SPRITES = List.of(Star()) ● update() ● getGraphicElement(): GameGraphicElement ● getCollider(): StarCollider ● bounce()

C SuperMushroom
<ul style="list-style-type: none"> ○ final POINTS_MARIO: int ○ final POINTS_SUPER_MARIO: int ◇ SPRITES_FOLDER: String ◇ collider: SuperMushroomCollider ◇ graphicElement: GameGraphicElement
<ul style="list-style-type: none"> ● SuperMushroom() ● getGraphicElement(): GameGraphicElement ● getCollider(): SuperMushroomCollider

C ChangeMarioColors
<ul style="list-style-type: none"> ○ <u>int: final</u>, : 50 ◇ <u>int: final</u> ◇ animator: ColorAnimator
<ul style="list-style-type: none"> ● ChangeMarioColors(: List<Map<Color, colorsMapping: Color>>, m: Mario) ● execute()

C Crouch
<ul style="list-style-type: none"> ○ <u>int: final</u> ◇ crouched: boolean ◇ mario: Mario
<ul style="list-style-type: none"> ● Crouch(m: Mario) ● execute() ● unCrouch(mario: Mario)

C DisappearSprite
<ul style="list-style-type: none"> ○ <u>int: final</u>, : 100 ◇ <u>int: final</u> ◇ disappear: int ◇ mario: Mario
<ul style="list-style-type: none"> ● DisappearSprite(m: Mario) ● execute()

C HorizontalMovement
<ul style="list-style-type: none"> ○ <u>int: final</u> ○ <u>int: final</u> ○ <u>float: final</u> ○ <u>float: final</u> ○ <u>int: final</u> ◇ mario: Mario
<ul style="list-style-type: none"> ● HorizontalMovement(m: Mario) ● execute() ◇ handleAirHorizontalMovement(m: Mario, currentDirection: Direction) ◇ handleGroundHorizontalMovement(m: Mario, currentDirection: Direction)

C ResolveHorizontalMovementDirection
<ul style="list-style-type: none"> ○ <u>int: final</u> ◇ mario: Mario
<ul style="list-style-type: none"> ● ResolveHorizontalMovementDirection(m: Mario) ● execute()

C ResolveSprite
<ul style="list-style-type: none"> ○ <u>int: final</u> ◇ <u>int: final</u>
<ul style="list-style-type: none"> ● ResolveSprite(m: Mario) ● execute()

I StrategyMarioAction
<ul style="list-style-type: none"> ● execute()

C ThrowFireBall
<ul style="list-style-type: none"> ○ <u>int: final</u> ○ <u>int: final</u> ○ <u>int: final</u> ◇ thrownBalls: int ◇ lastThrow: long ◇ mario: Mario
<ul style="list-style-type: none"> ● ThrowFireBall(m: Mario) ● execute() ◇ createFireBall() ● increaseAmmo()

C VerticalMovement
<ul style="list-style-type: none"> ○ <u>int: final</u> ○ <u>float: final</u> ○ <u>int: final</u>, : 8 ○ <u>int: final</u> ○ <u>float: final</u> ◇ mario: Mario
<ul style="list-style-type: none"> ● VerticalMovement(m: Mario) ● execute()

- String: final
- String: final
- final MARIO_STILL: String
- final MARIO_JUMP: String
- final MARIO_STOPPING: String
- int: final
- float: final
- ◇ jumpSpeed: float
- ◇ speedX: float
- ◇ speedY: float
- ◇ accelerationX: float
- ◇ falling: boolean
- ◇ starMarioColors: List<Map<Color, Color>>
- ◇ collider: MarioCollider
- ◇ graphicElement: GameGraphicElement
- ◇ actions: SortedSet<StrategyMarioAction>
- ◇ states: Map<Integer, CommandMarioStatus>
- ◇ stats: Stats
- ◇ movementDirection: Direction
- ◇ overrideSprite: boolean
- ◇ loaded: boolean

- ◇ List<Map<Color, Color>> COLOR_STAR_MARIO_COLORS = initStarColor(): final
- ◇ Color>> initStarColor(): List<Map<Color,
- final List<String> MARIO_WALKING = List.of(
- Mario(s: Stats)
- getCollider(): MarioCollider
- getGraphicElement(): GameGraphicElement
- update()
- land()
- die()
- isFalling(): boolean
- setFalling(j: boolean)
- addSpeed(dx: int, dy: int)
- getSpeedY(): float
- setSpeedY(speedY: float)
- getSpeedX(): float
- setSpeedX(speedX: float)
- addAction(action: StrategyMarioAction)
- removeAction(action: StrategyMarioAction)
- getAccelerationX(): float
- setAccelerationX(accelerationX: float)
- getMovementDirection(): Direction
- setMovementDirection(movementDirection: Direction)
- overrideSprite(): boolean
- setOverrideSprite(overrideSprite: boolean)
- Color>> getColorStarMarioColors(): List<Map<Color,
- setStarMarioColors(: List<Map<Color, initialColorStarMario: Color>>)
- modifyPoints(points: int)
- addLife()
- setState(state: CommandMarioStatus)
- removeState(state: CommandMarioStatus)
- setCollider(colliderToSet: MarioCollider): MarioCollider
- ◇ putColliderOnTop(newTopCollider: MarioCollider, bottomCollider: MarioCollider)
- ◇ swapCollider(oldCollider: MarioCollider, newCollider: MarioCollider)
- replaceCollider(c: MarioCollider)
- removeCollider(colliderToRemove: MarioCollider)

A

BaseMarioStatus

◇ mario: Mario

◇ previousCollider: MarioCollider

◇ newSpritesFolder: String

◇ String previousSpritesFolder

● BaseMarioStatus(m: Mario)

◇ swapSprites()

◇ revertSprites()

I

CommandMarioStatus

● setStatus()

● removeStatus()

C

FireMario

◇ String: final

◇ int: final

◇ fireBallThrower: ThrowFireBall

● FireMario(m: Mario)

● setStatus()

● removeStatus()

C

Invulnerable

◇ int: final

◇ invulnerableCollider: MarioCollider

◇ timer: Timer

◇ disappearSprite: DisappearSprite

● Invulnerable(m: Mario)

● setStatus()

● removeStatus()

C

StarMario

◇ int: final

◇ starCollider: MarioCollider

◇ colorChanger: ChangeMarioColors

◇ timer: Timer

● StarMario(m: Mario)

● setStatus()

● removeStatus()

C

SuperMario

◇ String: final

◇ int: final

◇ crouch: Crouch

● SuperMario(m: Mario)

● setStatus()

● removeStatus()

<div> <div></div> <div>LanguageConfiguration</div> </div>
<ul style="list-style-type: none"> ◊ <u>uniqueInstance: LanguageConfiguration</u> ◊ configuration: Properties ◊ <u>StringID: final</u> ◊ currLanguage: int ◊ language: Properties
<ul style="list-style-type: none"> ◆ LanguageConfiguration() ◆ <u>instance(): LanguageConfiguration</u> ◆ setLanguage(s: String) ◆ get(s: String): String ◆ nextLanguage()

<div> <div></div> <div>LevelReader</div> </div>
<ul style="list-style-type: none"> ◊ <u>int: final</u> ◊ <u>uniqueInstance: LevelReader</u> ◊ loaders: Map<Character, EntityLoader> ◊ <u>loadingStartingPoint: int</u> ◊ column: int ◊ row: int ◊ chunk: String
<ul style="list-style-type: none"> ■ LevelReader() <ul style="list-style-type: none"> ◆ <u>instance(): LevelReader</u> ◆ loadLevel(file: String) ◆ loadEntities(br: BufferedReader) ◆ loadScreen() ■ loadInvisibleColliders(lastChunkInScreen: int, windowHeight: int) ◆ loadUpdateablesEntitiesInScreen(lastChunkInScreen: int, loader: LoaderCollider) ◆ loadStats(stats: Stats) ◆ setColumn(i: int) <ul style="list-style-type: none"> ◆ getColumn(): int ◆ setRow(i: int) ◆ getRow(): int ◆ setChunk(s: String) ◆ getChunk(): String

<div> <div></div> <div>ObserverLevelStats</div> </div>

<div> <div></div> <div>RankingManager</div> </div>
<ul style="list-style-type: none"> ◊ <u>String: final</u> ◊ <u>highScores: List<ScoreEntry></u> static class ScoreEntry {
<ul style="list-style-type: none"> ◆ RankingManager() <ul style="list-style-type: none"> ◆ addScore(playerName: String, score: int) ■ loadHighScores() ■ saveHighScores() ◆ checkAndUpdateRanking(score: int): boolean ◆ getHighScores(): List<ScoreEntry>

<div> <div></div> <div>SingletonCollisionsEngine</div> </div>
<ul style="list-style-type: none"> ◊ <u>uniqueInstance: SingletonCollisionsEngine</u> ◊ chunks: List<List<Collider>> ◊ toUpdate: Set<Collider> ◊ currentCollider: Collider
<ul style="list-style-type: none"> ◆ SingletonCollisionsEngine() <ul style="list-style-type: none"> ◆ <u>instance(): SingletonCollisionsEngine</u> ◆ checkCollision(c1: Collider, c2: Collider, axis: Axis) ◆ update() ◆ checkCollisions(axis: Axis) ◆ checkChunk(axis: Axis, i: int) ◆ updateColliderBounds(previousBounds: Rectangle, c: Collider) ◆ calculateChunk(minX: int, maxX: int): int[] ◆ calculateChunk(bounds: Rectangle): int[] ◆ add(collider: Collider) ◆ removeFromChunks(bounds: Rectangle, c: Collider) ◆ remove(collider: Collider) ◆ getCollidersInRange(lowerBound: int, higherBound: int): Iterable<Collider> ◆ addToUpdate(c: Collider) ◆ reset() ◆ swap(toSwap: Collider, swapper: Collider)

<div> <div></div> <div>SingletonGame</div> </div>
<ul style="list-style-type: none"> ◊ <u>int: final</u> ◊ <u>int: final</u> ◊ <u>uniqueInstance: SingletonGame</u> ◊ toUpdateRegistry: Set<ObserverUpdateableEntity> ◊ keyStatus: Map<Integer, KeyStatus> ◊ stats: Stats ◊ executionQueue: Queue<Runnable> ◊ pause: boolean ◊ runGameLoop: boolean ◊ runGame: boolean ◊ debugging: boolean ◊ levels: String[] ◊ frames: long ◊ rankingManager: RankingManager
<ul style="list-style-type: none"> ◆ Object lock = new Object(); final ◆ toAddList = new ArrayList<>(); List<ObserverUpdateableEntity> ◆ toRemoveList = new ArrayList<>(); List<ObserverUpdateableEntity> ■ SingletonGame() <ul style="list-style-type: none"> ◆ <u>instance(): SingletonGame</u> ◆ registerToUpdate(e: ObserverUpdateableEntity) ◆ unregisterToUpdate(e: ObserverUpdateableEntity) ◆ getKeyStatus(key: int): KeyStatus ◆ runGame() <ul style="list-style-type: none"> ◆ loop() <ul style="list-style-type: none"> ◆ advanceLevel() ◆ checkGameOver() ◆ reloadGameStatus() ◆ checkRanking() ◆ showRanking() ◆ <u>main(args: String[])</u> ◆ resumeLoop() ◆ reload() ◆ resume() ◆ pause() ◆ togglePause() ◆ isDebugging(): boolean ◆ setDebugging(debugging: boolean) ◆ getLevelStats(): Stats ◆ getFrames(): long ◆ windowActivated(e: WindowEvent) ◆ windowClosed(e: WindowEvent) ◆ windowClosing(e: WindowEvent) ◆ windowDeactivated(e: WindowEvent) ◆ windowDeiconfied(e: WindowEvent) ◆ windowConfined(e: WindowEvent) ◆ windowOpened(e: WindowEvent) ◆ keyPressed(arg0: KeyEvent) ◆ keyReleased(arg0: KeyEvent) ◆ keyTyped(arg0: KeyEvent)

<div> <div></div> <div>SingletonGraphicEngine</div> </div>
<ul style="list-style-type: none"> ◊ <u>uniqueInstance: SingletonGraphicEngine</u> ◊ <u>String: final, 02: Comision</u> ◊ <u>Integer: final</u> ◊ <u>Integer: final</u> ◊ <u>Integer: final</u> ◊ <u>int: final</u> ◊ <u>int: final</u> ◊ <u>String: final</u> ◊ <u>StringID: final</u> ◊ mode: int ◊ position: int ◊ frame: JFrame ◊ panel: JLayeredPane ◊ backgrounds: GameGraphicElement[] ◊ <u>Set<GraphicElement>: ElementsOnScreen</u> ◊ dynamicElementsOnScreen: Set<TranslatableGraphicElement> ◊ toRedraw: List<GraphicElement> ◊ font: Font
<ul style="list-style-type: none"> ■ SingletonGraphicEngine() <ul style="list-style-type: none"> ◆ loadFont() ◆ initFrame() ◆ initBackgrounds() ◆ <u>instance(): SingletonGraphicEngine</u> ◆ drawFrame() ◆ add(e: GraphicElement) ◆ remove(e: GraphicElement) ◆ add(e: TranslatableGraphicElement) ◆ remove(e: TranslatableGraphicElement) ◆ <T extends GraphicElement> add(e: T, set: Set<T>) ◆ <T extends GraphicElement> remove(e: T, set: Set<T>) ◆ getPanelSize(): Dimension ◆ getPosition(): int ◆ setPosition(x: int) ◆ translate(dx: int) ◆ scrollScreen(velocity: int) { <ul style="list-style-type: none"> ◆ moveToBack(e: GraphicElement) ◆ moveToFront(e: GraphicElement) ◆ setDepth(e: GraphicElement, depth: Integer) ◆ reset() ◆ nextMode() ◆ reload() ◆ getMode(): String ◆ addToRedraw(graphicElement: GraphicElement) ◆ getFont(): Font ◆ setFont(font: Font) ◆ focusFrame()

<div> <div></div> <div>SingletonSoundManager</div> </div>
<ul style="list-style-type: none"> ◊ <u>uniqueInstance: SingletonSoundManager</u> ◊ clips: Map<String, Clip> ◊ soundPath: String
<ul style="list-style-type: none"> ■ SingletonSoundManager() <ul style="list-style-type: none"> ◆ <u>instance(): SingletonSoundManager</u> ◆ playSound(soundFile: String) ◆ playLoopingSound(soundFile: String) ◆ pauseAllSounds() ◆ resumeAllSounds() ◆ removeAllSounds()

<div> <div></div> <div>Stats</div> </div>
<ul style="list-style-type: none"> ◊ remainingTime: double ◊ score: int ◊ lives: int ◊ levelNumber: int ◊ lastActualization: long ◊ timer: Timer ◊ timerPaused: boolean ◊ initialTime: int ◊ initialLives: int ◊ observers: List<ObserverLevelStats>
<ul style="list-style-type: none"> ◆ Stats(initialTime: int, initialLives: int, numberLevel: int, scoreLevel: int) ◆ startTimer() ◆ addObserver(observer: ObserverLevelStats) ◆ notifyObserver() ◆ pauseTimer() ◆ resumeTimer() ◆ addLife() ◆ decreaseLives() ◆ modifyPoints(points: int) ◆ getLives() { : int <ul style="list-style-type: none"> ◆ getScore() { : int ◆ getRemainingTime() { : int ◆ getLevelNumber(): int ◆ setLevelNumber(levelNumber: int) ◆ advanceLevel() ◆ reset()

<div> <div></div> <div>StatsPanel</div> </div>
<ul style="list-style-type: none"> ◊ text: JLabel ◊ stat: JLabel
<ul style="list-style-type: none"> ◆ StatsPanel(f: Font) ◆ getText(): String ◆ setText(s: String) ◆ setStat(t: int)

A BaseGraphicElement
<ul style="list-style-type: none"> ◇ added: boolean
<ul style="list-style-type: none"> ● added(): boolean ● add() ● remove() ● setAdded(status: boolean)

A BaseTranslatableGraphicElement
<ul style="list-style-type: none"> ◇ label: JLabel ◇ bounds: Rectangle
<ul style="list-style-type: none"> ● BaseTranslatableGraphicElement() ● getComponent(): JComponent ● add() ● remove() ● translate(dx: int, dy: int) ● getPosition(): Point ● setPosition(x: int, y: int) ● redraw()

C FlyweightSpriteFactory
<ul style="list-style-type: none"> ◇ uniqueInstance: FlyweightSpriteFactory ◇ SPRITES_DIR: String ◇ sprites: Map<String, Map<String, ImageIcon>>
<ul style="list-style-type: none"> ■ FlyweightSpriteFactory() ● instance(): FlyweightSpriteFactory ● ImageIcon> getSprites(entity: String, mode: String): Map<String, ImageIcon> ■ ImageIcon> loadSprites(entity: String, mode: String): Map<String, ImageIcon> ■ loadImage(file: File): ImageIcon ● clearCache() ● hasSpritesLoaded(entity: String): boolean

C GameGraphicElement
<ul style="list-style-type: none"> ◇ entity: Entity ◇ sprite: ImageIcon ◇ currentSprite: String ◇ lastNotNullSprite: String ◇ colorRemap: Map<Color, Color> ◇ flipped: boolean ◇ iconUpdated: boolean ◇ sprites: Map<String, ImageIcon> ◇ folder: String
<ul style="list-style-type: none"> ● GameGraphicElement(e: Entity, folderPath: String) ● getEntity(): Entity ● getSpriteName(): String ● getSprite(): ImageIcon ● lastNotNullSpriteName(): String ● setSprite(s: String) ◇ forcefullyUpdateSprite(s: String) ● getComponent(): JLabel ● reload() ● redraw() ◇ loadSprites() ● setFolder(f: String) ● getFolder(): String ● isFlipped(): boolean ● Color> getColorRemap(): Map<Color, Color> ● setColorRemap: Map<Color, colorRemap: Color>) ● removeColorRemap() ● flipSprite() ● flipImage(image: BufferedImage): BufferedImage ● remapSpriteColor(: Map<Color, colorRemap: Color>, sprite: ImageIcon): ImageIcon ● iconToBufferedImage(icon: ImageIcon): BufferedImage

I GraphicElement
<ul style="list-style-type: none"> ● getComponent(): JComponent ● redraw() ● setAdded(status: boolean) ● added(): boolean ● add() ● remove() ● reload()

C RankingScreen
<ul style="list-style-type: none"> ◇ rankingManager: RankingManager
<ul style="list-style-type: none"> ● RankingScreen(ranking: RankingManager) ◇ initOverlay()

A ScreenOverlay
<ul style="list-style-type: none"> ○ Color: final ◇ panel: JPanel ◇ text: String ◇ label: JLabel ◇ font: Font ◇ bounds: Rectangle
<ul style="list-style-type: none"> ● ScreenOverlay(overlayText: String) ◇ initOverlay() ● getComponent(): JComponent ● redraw() ● reload() ● add()

C StatsBar
<ul style="list-style-type: none"> ◇ font: Font ◇ timePanel: StatsPanel ◇ livesPanel: StatsPanel ◇ levelPanel: StatsPanel ◇ scorePanel: StatsPanel ◇ mainPanel: JPanel ◇ levelStats: Stats
<ul style="list-style-type: none"> ● StatsBar(l: Stats) ● createPanel(font: Font): StatsPanel ● addElement(element: StatsPanel) ● redraw() ● getComponent(): JComponent ● getFont(): Font ● onStatsChanged() ● reload()

C TemporaryScreenOverlay
<ul style="list-style-type: none"> ◇ timeout: int
<ul style="list-style-type: none"> ● TemporaryScreenOverlay(overlayText: String, timeout: int) ● add()

C TextLabel
<ul style="list-style-type: none"> ◇ data: String ◇ font: Font
<ul style="list-style-type: none"> ● TextLabel(s: String) ● redraw() ● reload() ● getComponent(): JComponent

I TranslatableGraphicElement
<ul style="list-style-type: none"> ● translate(dx: int, dy: int) ● getPosition(): Point ● setPosition(x: int, y: int)

E

Axis

}package utils;

abstract class BasePrioritizable {

priority: int

getPriority(): int

setPriority(p: int)

E

Direction

opposite(d1: Direction): Direction

sum(d1: Direction, d2: Direction): Direction

horizontalDirectionFromSign(d: int): Direction

signFromDirection(d1: Direction): int

verticalDirectionFromSign(d: int): Direction

E

KeyStatus

I

Prioritizable

getPriority(): int

setPriority(p: int)

C

PriorityComparator

compare(arg0: Prioritizable, arg1: Prioritizable): int

C

Vector2D

start: Point

end: Point

xComponent: double

yComponent: double

(s: Point, e: Point): Vector2D

calculateComponents()

getStart(): Point

getEnd(): Point

sum(v: Vector2D): Vector2D

grow(dx: int, dy: int)

translate(dx: int, dy: int)

getXComponent(): double

getYComponent(): double

clone(): Vector2D

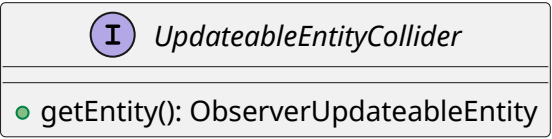
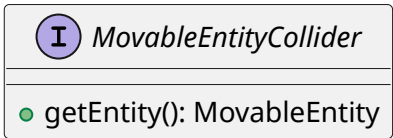
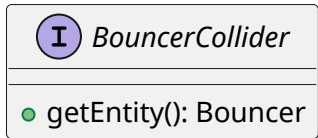
toString(): String

C FireBallCollision
◇ collider: FireBallCollider
<ul style="list-style-type: none">● FireBallCollision(c: FireBallCollider)● getCollider(): FireBallCollider● collide(c: BlockCollider, a: Axis)● collide(c: ScreenDisplacementCollider, a: Axis)● collide(c: ScreenBorderCollider, a: Axis)● collide(c: SpinyCollider, a: Axis)● collide(c: KoopaTroopaCollider, a: Axis)● collide(c: SuperMushroomCollider, a: Axis)● collide(c: GoombaCollider, a: Axis)● collide(c: LoaderCollider, a: Axis)● collide(c: BrickCollider, a: Axis)● collide(c: PipeCollider, a: Axis)● collide(c: QuestionBlockCollider, a: Axis)● collide(c: CoinCollider, a: Axis)● collide(c: GraphicUnloaderCollider, a: Axis)● collide(c: DeleterCollider, a: Axis)● collide(c: LakituCollider, a: Axis)● collide(c: BuzzyBeetleCollider, a: Axis)● collide(c: PiranhaPlantCollider, a: Axis)● collide(c: FireFlowerCollider, a: Axis)● collide(c: StarMarioCollider, a: Axis)● collide(c: SuperMarioCollider, a: Axis)● collide(c: DefaultMarioCollider, a: Axis)● collide(c: GreenMushroomCollider, a: Axis)● collide(c: EmptyBlockCollider, a: Axis)● collide(c: LevelEndCollider, a: Axis)● collide(c: InvulnerableCollider, a: Axis)● collide(c: FlagPoleCollider, a: Axis)● collide(c: FireMarioCollider, a: Axis)● collide(c: FireBallCollider, a: Axis)● collide(c: ModeSwitcherCollider, a: Axis)● collide(c: LanguageSwitcherCollider, a: Axis)● collide(c: StarCollider, a: Axis)● collide(c: RankingShowCollider, a: Axis)

I BouncerCollision
<ul style="list-style-type: none">● getCollider(): BouncerCollider

I MovableEntityCollision
<ul style="list-style-type: none">● getCollider(): MovableEntityCollider

I UpdateableEntityCollision
<ul style="list-style-type: none">● getCollider(): UpdateableEntityCollider



C DefaultMarioCollider
<ul style="list-style-type: none"> o <u>int: final</u>
<ul style="list-style-type: none"> ● DefaultMarioCollider(m: Mario, b: Rectangle) ● getCollision(): DefaultMarioCollision ● recieveCollision(c: VisitorCollision, a: Axis)

C FireMarioCollider
<ul style="list-style-type: none"> o <u>int: final</u>
<ul style="list-style-type: none"> ● FireMarioCollider(m: Mario, b: Rectangle, f: FireMario) ● getCollision(): FireMarioCollision ● recieveCollision(c: VisitorCollision, a: Axis)

C InvulnerableCollider
<ul style="list-style-type: none"> ◇ mario: Mario o <u>int: final</u>
<ul style="list-style-type: none"> ● InvulnerableCollider(m: Mario) ● getEntity(): Mario ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): InvulnerableCollision

A MarioCollider
<ul style="list-style-type: none"> ◇ mario: Mario ◇ baseCollider: MarioCollider ◇ colliderOnTop: MarioCollider ◇ associatedState: CommandMarioStatus ◇ priority: int
<ul style="list-style-type: none"> ● MarioCollider(b: Rectangle) ● getEntity(): Mario ● MarioCollision getCollision(): abstract ● getBaseCollider(): MarioCollider ● getAssociatedState(): CommandMarioStatus ● setBaseCollider(c: MarioCollider) ● getPriority(): int ● setPriority(priority: int) ● getColliderOnTop(): MarioCollider ● setColliderOnTop(collider: MarioCollider)

C StarMarioCollider
<ul style="list-style-type: none"> ◇ mario: Mario o <u>int: final</u>
<ul style="list-style-type: none"> ● StarMarioCollider(m: Mario, starMario: StarMario) ● getEntity(): Mario ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): StarMarioCollision

C SuperMarioCollider
<ul style="list-style-type: none"> o <u>int: final</u>
<ul style="list-style-type: none"> ● SuperMarioCollider(m: Mario, b: Rectangle, s: SuperMario) ● getCollision(): SuperMarioCollision ● recieveCollision(c: VisitorCollision, a: Axis)

A

BaseLoader

◆ positionCollider (e: Entity, row: int, column: int)

C

BlockLoader

● load(lr: LevelReader)

C

BrickLoader

● load(lr: LevelReader)

C

BuzzyBeetleLoader

● load(lr: LevelReader)

C

CastleLoader

● load(lr: LevelReader)

C

CoinLoader

● load(lr: LevelReader)

C

ConfigurationBlockLoader

● load(lr: LevelReader)

C

EmptyBlockLoader

● load(lr: LevelReader)

I

EntityLoader

● load(lr: LevelReader)

C

FireFlowerLoader

● load(lr: LevelReader)

C

FlagPoleLoader

● load(lr: LevelReader)

C

GoombaLoader

● load(lr: LevelReader)

C

GreenMushroomLoader

● load(lr: LevelReader)

C

KoopaTroopaLoader

● load(lr: LevelReader)

C

LakituLoader

● load(lr: LevelReader)

C

MarioLoader

● load(lr: LevelReader)

C

PipeLoader

● load(lr: LevelReader)

C

PiranhaPlantLoader

C

QuestionBlockLoader

● load(lr: LevelReader)

C

SpinyLoader

● load(lr: LevelReader)

C


StarLoader


● load(lr: LevelReader)


C


SuperMushroomLoader


● load(lr: LevelReader)

 BlockCollider
◇ block: Block
<ul style="list-style-type: none"> ● BlockCollider(e: Block, b: Rectangle) ● getEntity(): Block ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): BlockCollision

 BrickCollider
◇ brick: Brick
<ul style="list-style-type: none"> ● BrickCollider(brick2: Brick, b: Rectangle) ● getEntity(): Brick ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): BrickCollision ● handleVerticalCollision(m: MarioCollision) ● handleVerticalCollision(m: SuperMarioCollision) ◇ fallBackIntoPlace(displacement: int)

 PipeCollider
◇ pipe: Pipe
<ul style="list-style-type: none"> ● PipeCollider(e: Pipe, b: Rectangle) ● getEntity(): Pipe ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): PipeCollision ● handleVerticalCollision(p: PiranhaPlantCollision) ● handleHorizontalCollision(p: PiranhaPlantCollision)

 QuestionBlockCollider
◇ questionBlock: QuestionBlock
<ul style="list-style-type: none"> ● QuestionBlockCollider(e: QuestionBlock, b: Rectangle) ● getEntity(): QuestionBlock ● recieveCollision(c: VisitorCollision, a: Axis) ● getCollision(): QuestionBlockCollision ● handleVerticalCollision(m: MarioCollision) ● handleVerticalCollision(m: SuperMarioCollision) ◇ marioVerticalCollision(p: PowerUp, m: MarioCollision) ◇ fallBackIntoPlace(displacement: int)

 SolidCollider
<ul style="list-style-type: none"> ● SolidCollider(b: Rectangle) ● handleHorizontalCollision(e: UpdateableEntityCollision) ● handleVerticalCollision(e: UpdateableEntityCollision) ● handleVerticalCollision(m: MovableEntityCollision) ● handleHorizontalCollision(m: MarioCollision) ● handleVerticalCollision(m: MarioCollision) ● handleHorizontalCollision(e: EnemyCollision) ● handleVerticalCollision(e: EnemyCollision) ● handleHorizontalCollision(p: PowerUpCollision) ● handleVerticalCollision(s: BouncerCollision) ● handleHorizontalCollision(p: FireBallCollision) ● displaceHorizontally(c: Collider): int ● displaceVertically(c: Collider): int ◇ fallBackIntoPlace(displacement: int)

A

Body

•

setSpritesFolder(folder: String)

•

setSprite(s: String)

•

adjustGraphicElementOnChange(newSprite: ImageIcon, previousSprite: ImageIcon)

•

adjustColliderToGraphicElement()

•

translate(dx: int, dy: int)

•

displaceHorizontally(c: Collider): int

•

displaceVertically(c: Collider): int

•

spawnEntity(e: ObserverUpdateableEntity, diffX: int, diffY: int)

C

ConfigurationBlock

◇

SPRITES_FOLDER: String

○

int: final

◇

collider: SolidCollider

◇

graphicElement: GameGraphicElement

◇

text: TextLabel

◇

animator: MovementAnimator

•

final List<String> ANIMATED_SPRITES = List.of(

•

ConfigurationBlock(s: char)

•

load()

•

getGraphicElement(): GameGraphicElement

•

getCollider(): SolidCollider

•

update()

C

EmptyBlock

○

final int POINTS = -15

◇

collider: EmptyBlockCollider

◇

graphicElement: GameGraphicElement

•

EmptyBlock()

•

getGraphicElement(): GameGraphicElement

•

getCollider(): EmptyBlockCollider

I

Entity

•

getGraphicElement(): GameGraphicElement

•

getCollider(): Collider

C

FlagPole

◇

SPRITES_FOLDER: String

◇

collider: FlagPoleCollider

◇

graphicElement: GameGraphicElement

•

FlagPole()

•

getGraphicElement(): GameGraphicElement

•

getCollider(): FlagPoleCollider