Diseño de sistemas

TRABAJO PRÁCTICO 5: STARSHIP GAME (IMPLEMENTACIÓN)

Perez Molina, Tomás

6 de noviembre de 2018

Índice

1. Consigna

1.1. Definición

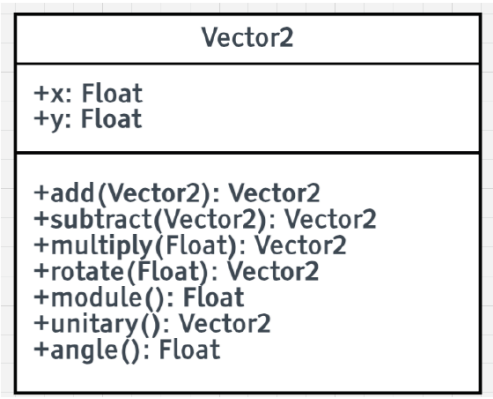
Trabajando para un sitio de juegos llamado “Jug.ar” tu director de desarrollo decide que se debe crear un juego con naves espaciales ya que dentro de la amplia gama de juegos que ofrecen no hay ninguno con características. La tarea se delega en tu equipo y como primer paso deciden armar un prototipo de un juego con naves. En esta oportunidad, la tarea de llevar a cabo el diseño esta en tus manos.

Post “torbellino de ideas” (brainstorming) tu equipo y vos deciden diseñar un juego en el que cada jugador tenga que maniobrar su nave mientras esquiva y destruye meteoritos que vuelan por el espacio. Adicionalmente las naves se podrán destruir entre ellos y tendrán una cantidad de vidas, también serán destruidas por los asteroides con los que se estrellen. También deciden que al ser un prototipo debe ser lo mas adaptable posible por lo que concuerdan en la aplicación del patron MVC[1]. Para una primera versión se limitan los requerimientos para que el juego pueda:

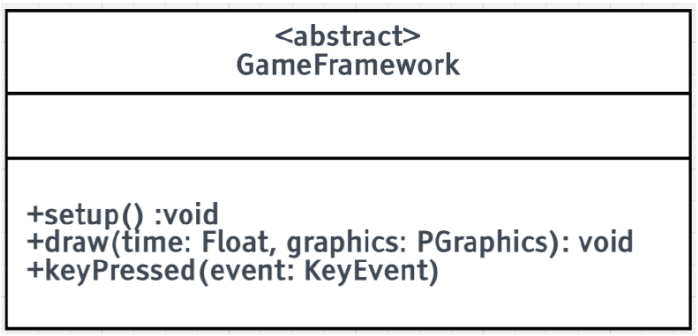
- El juego sera en 2D
- Cada jugador podrá maniobrar su nave, a travez de una serie de teclas configurables (acelerar, frenar, girar). Las naves solo se pueden mover dentro del area correspondiente a la pantalla.
- Cada nave tiene la capacidad de disparar, el disparo depende del arma que tenga la nave, y puede disparar una o mas balas de distintos tamaños. Al impactar en otra nave, o en un asteroide lo destruyen y le asignan al jugador la cantidad de puntos correcta
- El escenario presenta asteroides de distintos tamaños que se mueven por la escena, estos aparecen desde los limites de la pantalla de forma aleatoria. (En principio se decide no manejar la coalición entre estos).

La empresa ya dispone de algunas herramientas que se pueden usar para ahorrar tiempo en el diseño de este prototipo, aunque no es obligatorio:

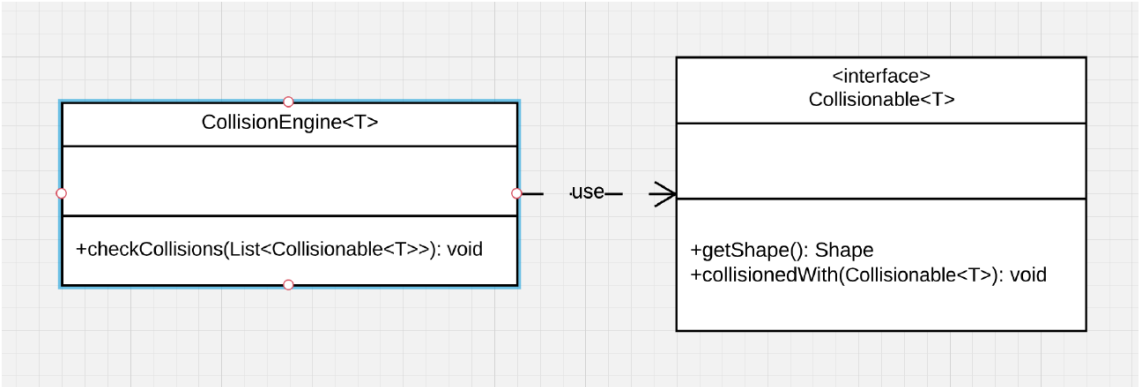
La representación de un vector de dos dimensiones con sus operaciones básicas:



Un framework que se encarga de levantar la aplicación, el repintado constante de la interfaz gráfica y manejar los eventos de teclado. El método ‘draw’ se llama constantemente (A una frecuencia aproximada de 60 veces por segundo), con el finde poder actualizar el estado del juego y la interfaz gráfica. Como argumentos recibe tiempo que paso entre cada dibujo, así como también una herramienta que permite dibujar desde figuras básicas hasta el uso de imágenes, denominada PGraphics.



Por ultimo existe un motor de calculo de colisiones que ya se a utilizado anteriormente basado la clase java.awt.Shape[2] de java.



[1] MVC = patron Model-View- Controller el cual establece que debe haber una clara separación entre dichas capas dentro de un sistema. El Modelo representara nuestro Dominio de entidades y su logica de negocio, La Vista será quien se encargue de mostrar los distintos estados o resultados obtenidos a partir del modelo y por ultimo, el Control sera quien se ocupe de orquestar la comunicación entre Modelo y Vista.

[2] https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html

1.2. Entrega

1. Implementar el juego Starships a partir del diseño propuesto, realizando los cambios necesarios
2. Modificar el UML original para reflejar los cambios realizados.

2. Cambios

2.1. Aclaraciones

Menu Los menu se limitaron a un menú principal y a un menú de pausa, lo que simplifico la interfaz para ser simplemente un **ScreenController**.

Router controller **BaseController** fue reemplazado por **RouterController**, que permite que las Screens tengan mas control sobre el flujo de la UI. Además ahora cada screen solo tiene un router, en vez de ser un **Observable <ControllerCreationObserver>**, ya que multiples routers no deberian ser notificados de los mismos cambios al mismo tiempo.

Configuración Se implementó **INIControlConfigurator** para configurar el juego a partir de un archivo *.ini. Que permite la definición del control de la UI, así también como la de todos los jugadores. La cantidad de jugadores siendo solo limitada por el tamaño del teclado.

ProcessingKeyEventAdapter Debido al manejo de KeyEvents de Processing, fue necesario incluir un *adapter* que permitiera la correcta funcionalidad del KeyEventHandler propuesto en el diseño.

ShipSpawner Para soportar la capacidad de respawnear naves ya en juego, se agrego la interfaz **ShipSpawner** que lo permite, de esta manera se pueden dar multiple cantidad de vidas a cada jugador.

Collisiones Se reemplazaron los **Collision** por **Collider** que implementan la interfaz **Collisionable <Collider>**, de esta manera los **GameEntity** no deben implementar **Collisionable**

Bounds Al notar que tanto los bordes del mundo, como los de la pantalla debian ser conocidos por distintas partes de la aplicación, se implementó la clase **Bounds** para poder abstraer ese concepto y agregar alguno métodos útiles.

PowerUps Para permitir distintas vistas para los distintos **PowerUp**, se agrego un **PowerUpVisitor** para poder recuperar el tipo al momento de dibujar.

Plane Se agrego la capacidad de dibujar texto, y se cambió la interfaz de **DrawColors** para recibir un java.awt.Color

2.2. Patrones de diseño

Adapter Se expandió la utilización de *adapter* para poder utilizar la API de Processing con el diseño propuesto anteriormente.

Visitor Se agregó un *visitor* de **PowerUp** para poder dibujar cada uno de diferente manera.

2.3. Diagramas de clases

Se dividió el diagrama de clases en cuatro partes para disminuir el tamaño de los gráficos y que sea más facil la visualización.

notifyBack()
..... c = controllers.pop()
..... c.deregister(handler)

notifyNewController(controller)
..... controllers.push(controller)
..... controller.register(handler)

update()
.....controllers.peek().update()

render(plane)
.....controllers.peek().render(plane)

«interface»
ControllerRouter

+ back(): void
+ goToStart(): void
+ newController(ScreenController): void

BaseController

- configuration: BaseControllerConfiguration
- controllers: Stack<ScreenController>
- handler: KeyEventHandler
- initialScreen: ScreenController

+ goToStart(): void
+ notifyBack(): void
+ notifyNewController(ScreenController): void
+ render(Plane): void
+ update(): void

«interface»
ScreenController

+ addRouter(ControllerRouter): void
+ deregister(KeyEventHandler): void
+ register(KeyEventHandler): void
+ removeRouter(): void
+ render(Plane): void
+ update(): void

GameOverScreen

- backKeyName: String
- highScorePlayer: Player
- screenBounds: Bounds

+ addRouter(ControllerRouter): void
+ deregister(KeyEventHandler): void
+ register(KeyEventHandler): void
+ removeRouter(): void
+ render(Plane): void
+ update(): void

PauseScreen

- backKeyName: String
- gameScreen: ScreenController
- router: ControllerRouter
- screenBounds: Bounds
- selectConfiguration: SelectConfiguration
- selectKeyName: String

+ addRouter(ControllerRouter): void
+ deregister(KeyEventHandler): void
+ register(KeyEventHandler): void
+ removeRouter(): void
+ render(Plane): void
+ update(): void

BaseControllerConfiguration

+ backKey: int

StartScreen

- gamePausePairGen: Producer<GamePausePair>
- router: ControllerRouter
- screenBounds: Bounds
- selectConfiguration: SelectConfiguration
- selectKeyName: String

+ addRouter(ControllerRouter): void
+ deregister(KeyEventHandler): void
+ register(KeyEventHandler): void
+ removeRouter(): void
+ render(Plane): void
+ update(): void

SelectConfiguration

+ selectKey: int

«interface»
AnimationController

+ render(Plane): void
+ update(): void

GameFramework

+ draw(float, PGraphics, Set<int>): void
+ keyPressed(KeyEvent): void
+ keyReleased(KeyEvent): void
+ setup(): void

«interface»
GameplayController

+ addShip(Ship, Color): void
+ render(Plane): void
+ update(): void

StarshipGame

- baseController: AnimationController
- configurator: ControlConfigurator
- gameBounds: Bounds
- handler: ProcessingKeyEventAdapter
- screenBounds: Bounds

+ draw(float, PGraphics): void
+ keyPressed(KeyEvent): void
+ setup(): void

GameController

- destroyVisitor: EntityVisitor<int>
- gameOverScreenGen: Producer<ScreenController>
- gameplayController: GameplayController
- hitVisitor: EntityVisitor<int>
- hudController: HUDController
- playerShips: Map<UUID, Player>
- router: ControllerRouter
- shipColors: Color [0..*]
- shipSpawnController: ShipSpawnController

+ addRouter(ControllerRouter): void
+ deregister(KeyEventHandler): void
+ register(KeyEventHandler): void
+ removeRouter(): void
+ render(Plane): void
+ update(): void

ControlConfiguration

- baseControl: BaseControllerConfiguration
- selectControl: SelectConfiguration
- shipControls: ShipControllerConfiguration [0..*]
- shipSpawn: ShipSpawnControllerConfiguration

«interface»
HUDController

+ addPlayer(Drawable): void
+ render(Plane): void
+ update(): void

StarShipHUDController

- gameBounds: Bounds
- playerViews: Drawable
- screenBounds: Bounds

+ render(Plane): void
+ update(): void

«interface»
KeyEventHandler

+ addObserver(int, KeyEventObserver): void
+ keyPressed(KeyEvent): void
+ removeObserver(int): void

«interface»
KeyEventController

+ deregister(KeyEventHandler): void
+ register(KeyEventHandler): void

«interface»
ShipSpawnController

+ deregister(KeyEventHandler): void
+ getNew(): Ship [ ]
+ notifySpawnShip(): void
+ register(KeyEventHandler): void
+ respawnShip(Ship): Ship

ConfigurableShipSpawnController

- configuration: ShipSpawnControllerConfiguration
- handler: KeyEventHandler
- shipConfigurations: ShipControllerConfiguration [0..*]
- shipControllers: ShipController [0..*]
- spawner: Spawner<Ship>
- spawnLocations: Vector2D

+ deregister(KeyEventHandler): void
+ getNew(): Ship [ ]
+ notifySpawnShip(): void
+ register(KeyEventHandler): void
+ respawnShip(Ship): Ship

ShipSpawnControllerConfiguration

+ spawnShipKey: int

«interface»
KeyEventObserver

+ notifyKeyPressed(): void

KeyEventHandler

- observers: Map<int, KeyEventObserver>

+ addObserver(int, KeyEventObserver): void
+ keyPressed(KeyEvent): void
+ removeObserver(int): void

«interface»
ShipController

+ deregister(KeyEventHandler): void
+ getShip(): ControllableShip
+ notifyBackward(): void
+ notifyForward(): void
+ notifyRotateLeft(): void
+ notifyRotateRight(): void
+ notifyShoot(): void
+ register(KeyEventHandler): void
+ setShip(ControllableShip): void

ConfigurableShipController

- configuration: ShipControllerConfiguration
- ship: ControllableShip

+ deregister(): void
+ getShip(): ControllableShip
+ notifyBackward(): void
+ notifyForward(): void
+ notifyRotateLeft(): void
+ notifyRotateRight(): void
+ notifyShoot(): void
+ register(KeyEventHandler): void
+ setShip(ControllableShip): void

ShipControllerConfiguration

+ backwardKey: int
+ forwardKey: int
+ leftKey: int
+ rightKey: int
+ shootKey: int

ProcessingKeyEventAdapter

- handler: KeyEventHandler
- pressed: Set<int>

+ keyPressed(KeyEvent): void
+ keyReleased(KeyEvent): void
+ notifyHandler(): void

Accelerateable
Rotateable
Shooter

«interface»
ControllableShip

+ applyAcceleration(float): void
+ applyRotation(float): void
+ shoot(): void

«interface»
ControlConfigurator

+ getConfiguration(): ControlConfigurator

INIControlConfigurator

- default: ControlConfiguration
- fileName: String

+ getConfiguration(): GameConfiguration

GameConfiguration

- controlConfiguration: ControlConfiguration
- gameplayConfiguration: GameplayConfiguration
- hudConfiguration: HUDConfiguration
- paintConfiguration: PaintConfiguration

HUDConfiguration

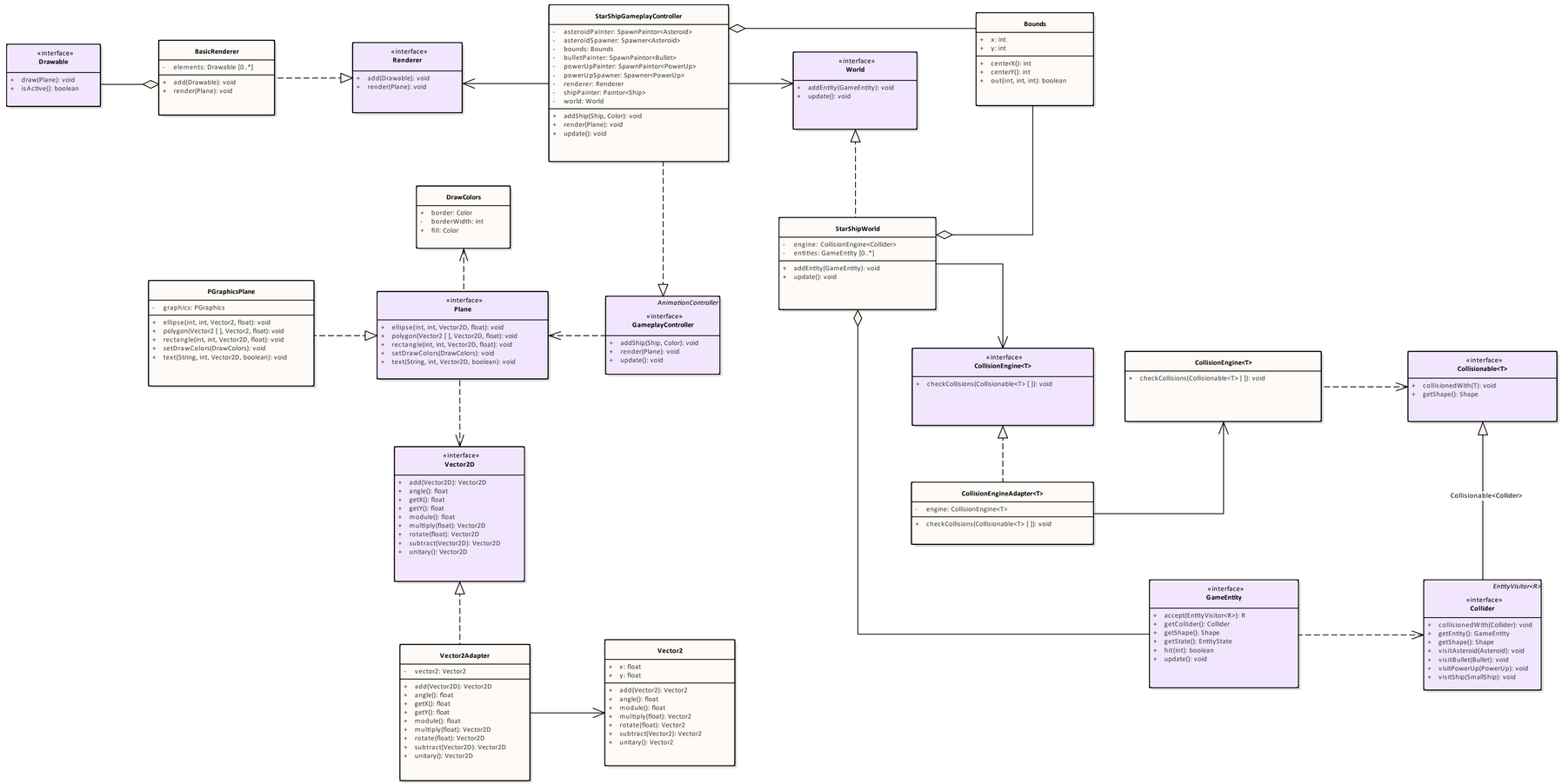
- backKeyName: String
- selectKeyName: String

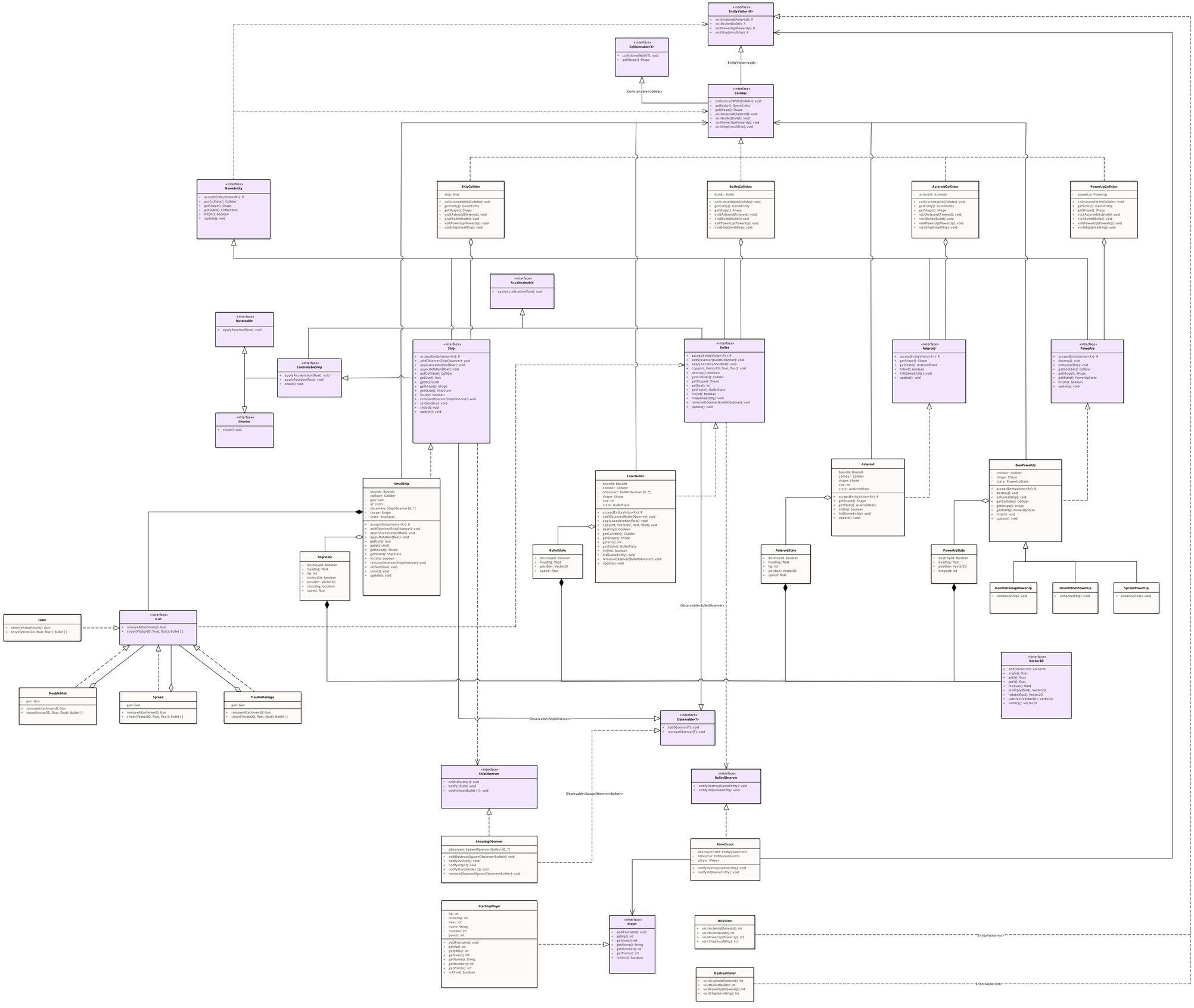
PaintConfiguration

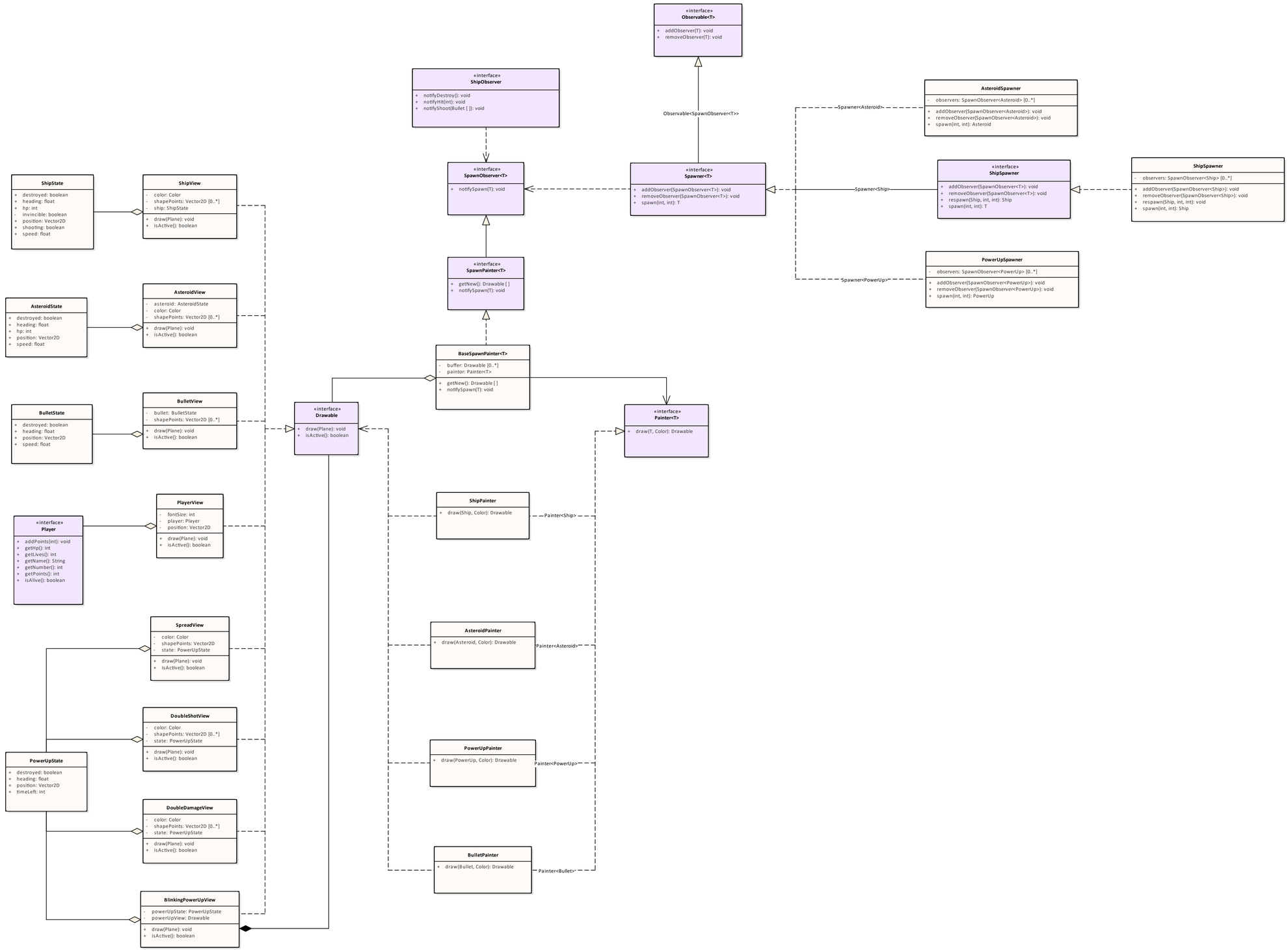
- shipColors: Color [0..*]

GameplayConfiguration

- bounds: Bounds
- shipSpawnLocations: Vector2D [0..*]


«interface»
Drawable
+ draw(Plane): void
+ isActive(): boolean
BasicRenderer
- elements: Drawable [0..*]
+ add(Drawable): void
+ render(Plane): void
«interface»
Renderer
+ add(Drawable): void
+ render(Plane): void
StarShipGameplayController
- asteroidPainter: SpawnPaintor<Asteroid>
- asteroidSpawner: Spawner<Asteroid>
- bounds: Bounds
- bulletPainter: SpawnPaintor<Bullet>
- powerUpPainter: SpawnPaintor<PowerUp>
- powerUpSpawner: Spawner<PowerUp>
- renderer: Renderer
- shipPainter: Paintor<Ship>
- world: World
+ addShip(Ship, Color): void
+ render(Plane): void
+ update(): void
«interface»
World
+ addEntity(GameEntity): void
+ update(): void
Bounds
+ x: int
+ y: int
+ centerX(): int
+ centerY(): int
+ out(int, int, int): boolean
DrawColors
+ border: Color
- borderWidth: int
+ fill: Color
StarShipWorld
- engine: CollisionEngine<Collider>
- entities: GameEntity [0..*]
+ addEntity(GameEntity): void
+ update(): void
PGraphicsPlane
- graphics: PGraphics
+ ellipse(int, int, Vector2, float): void
+ polygon(Vector2 [ ], Vector2, float): void
+ rectangle(int, int, Vector2D, float): void
+ setDrawColors(DrawColors): void
+ text(String, int, Vector2D, boolean): void
«interface»
Plane
+ ellipse(int, int, Vector2D, float): void
+ polygon(Vector2 [ ], Vector2D, float): void
+ rectangle(int, int, Vector2D, float): void
+ setDrawColors(DrawColors): void
+ text(String, int, Vector2D, boolean): void
AnimationController
«interface»
GameplayController
+ addShip(Ship, Color): void
+ render(Plane): void
+ update(): void
«interface»
CollisionEngine<T>
+ checkCollisions(Collisionable<T> [ ]): void
CollisionEngine<T>
+ checkCollisions(Collisionable<T> [ ]): void
«interface»
Collisionable<T>
+ collisionedWith(T): void
+ getShape(): Shape
CollisionEngineAdapter<T>
- engine: CollisionEngine<T>
+ checkCollisions(Collisionable<T> [ ]): void
Collisionable<Collider>
«interface»
Vector2D
+ add(Vector2D): Vector2D
+ angle(): float
+ getX(): float
+ getY(): float
+ module(): float
+ multiply(float): Vector2D
+ rotate(float): Vector2D
+ subtract(Vector2D): Vector2D
+ unitary(): Vector2D
Vector2Adapter
- vector2: Vector2
+ add(Vector2D): Vector2D
+ angle(): float
+ getX(): float
+ getY(): float
+ module(): float
+ multiply(float): Vector2D
+ rotate(float): Vector2D
+ subtract(Vector2D): Vector2D
+ unitary(): Vector2D
Vector2
+ x: float
+ y: float
+ add(Vector2): Vector2
+ angle(): float
+ module(): float
+ multiply(float): Vector2
+ rotate(float): Vector2
+ subtract(Vector2): Vector2
+ unitary(): Vector2
«interface»
GameEntity
+ accept(EntityVisitor<R>): R
+ getCollider(): Collider
+ getShape(): Shape
+ getState(): EntityState
+ hit(int): boolean
+ update(): void
EntityVisitor<R>
«interface»
Collider
+ collisionedWith(Collider): void
+ getEntity(): GameEntity
+ getShape(): Shape
+ visitAsteroid(Asteroid): void
+ visitBullet(Bullet): void
+ visitPowerUp(PowerUp): void
+ visitShip(SmallShip): void


«interface»
EntityVisitor<R>
+ visitAsteroid(Asteroid): R
+ visitBullet(Bullet): R
+ visitPowerUp(PowerUp): R
+ visitShip(SmallShip): R
«interface»
Collisionable<T>
+ collisionedWith(T): void
+ getShape(): Shape
EntityVisitor<void>
Collisionable<Collider>
«interface»
Collider
+ collisionedWith(Collider): void
+ getEntity(): GameEntity
+ getShape(): Shape
+ visitAsteroid(Asteroid): void
+ visitBullet(Bullet): void
+ visitPowerUp(PowerUp): void
+ visitShip(SmallShip): void
«interface»
GameEntity
+ accept(EntityVisitor<R>): R
+ getCollider(): Collider
+ getShape(): Shape
+ getState(): EntityState
+ hit(int): boolean
+ update(): void
ShipCollider
- ship: Ship
+ collisionedWith(Collider): void
+ getEntity(): GameEntity
+ getShape(): Shape
+ visitAsteroid(Asteroid): void
+ visitBullet(Bullet): void
+ visitPowerUp(PowerUp): void
+ visitShip(SmallShip): void
BulletCollision
- bullet: Bullet
+ collisionedWith(Collider): void
+ getEntity(): GameEntity
+ getShape(): Shape
+ visitAsteroid(Asteroid): void
+ visitBullet(Bullet): void
+ visitPowerUp(PowerUp): void
+ visitShip(SmallShip): void
AsteroidCollision
- asteroid: Asteroid
+ collisionedWith(Collider): void
+ getEntity(): GameEntity
+ getShape(): Shape
+ visitAsteroid(Asteroid): void
+ visitBullet(Bullet): void
+ visitPowerUp(PowerUp): void
+ visitShip(SmallShip): void
PowerUpCollision
- powerUp: PowerUp
+ collisionedWith(Collider): void
+ getEntity(): GameEntity
+ getShape(): Shape
+ visitAsteroid(Asteroid): void
+ visitBullet(Bullet): void
+ visitPowerUp(PowerUp): void
+ visitShip(SmallShip): void
«interface»
Acceleratable
+ applyAcceleration(float): void
«interface»
Rotateable
+ applyRotation(float): void
«interface»
ControllableShip
+ applyAcceleration(float): void
+ applyRotation(float): void
+ shoot(): void
«interface»
Shooter
+ shoot(): void
«interface»
Ship
+ accept(EntityVisitor<R>): R
+ addObserver(ShipObserver): void
+ applyAcceleration(float): void
+ applyRotation(float): void
+ getCollider(): Collider
+ getGun(): Gun
+ getId(): UUID
+ getShape(): Shape
+ getState(): ShipState
+ hit(int): boolean
+ removeObserver(ShipObserver): void
+ setGun(Gun): void
+ shoot(): void
+ update(): void
«interface»
Bullet
+ accept(EntityVisitor<R>): R
+ addObserver(BulletObserver): void
+ applyAcceleration(float): void
+ copy(int, Vector2D, float, float): void
+ destroy(): boolean
+ getCollider(): Collider
+ getShape(): Shape
+ getSize(): int
+ getState(): BulletState
+ hit(int): boolean
+ hit(GameEntity): void
+ removeObserver(BulletObserver): void
+ update(): void
«interface»
Asteroid
+ accept(EntityVisitor<R>): R
+ getShape(): Shape
+ getState(): AsteroidState
+ hit(int): boolean
+ hit(GameEntity): void
+ update(): void
«interface»
PowerUp
+ accept(EntityVisitor<R>): R
+ destroy(): void
+ enhance(Ship): void
+ getCollider(): Collider
+ getShape(): Shape
+ getState(): PowerUpState
+ hit(int): boolean
+ update(): void
SmallShip
- bounds: Bounds
- collider: Collider
- gun: Gun
- id: UUID
- observers: ShipObserver [0..*]
- shape: Shape
- state: ShipState
+ accept(EntityVisitor<R>): R
+ addObserver(ShipObserver): void
+ applyAcceleration(float): void
+ applyRotation(float): void
+ getGun(): Gun
+ getId(): UUID
+ getShape(): Shape
+ getState(): ShipState
+ hit(int): boolean
+ removeObserver(ShipObserver): void
+ setGun(Gun): void
+ shoot(): void
+ update(): void
ShipState
+ destroyed: boolean
+ heading: float
+ hp: int
+ invincible: boolean
+ position: Vector2D
+ shooting: boolean
+ speed: float
LaserBullet
- bounds: Bounds
- collider: Collider
- observers: BulletObserver [0..*]
- shape: Shape
- size: int
- state: BulletState
+ accept(EntityVisitor<R>): R
+ addObserver(BulletObserver): void
+ applyAcceleration(float): void
+ copy(int, Vector2D, float, float): void
+ destroy(): boolean
+ getCollider(): Collider
+ getShape(): Shape
+ getSize(): int
+ getState(): BulletState
+ hit(int): boolean
+ hit(GameEntity): void
+ removeObserver(BulletObserver): void
+ update(): void
BulletState
+ destroyed: boolean
+ heading: float
+ position: Vector2D
+ speed: float
AsteroidState
+ destroyed: boolean
+ heading: float
+ hp: int
+ position: Vector2D
+ speed: float
PowerUpState
+ destroyed: boolean
+ heading: float
+ position: Vector2D
+ timeLeft: int
GunPowerUp
- collider: Collider
- shape: Shape
- state: PowerUpState
+ accept(EntityVisitor<R>): R
+ destroy(): void
+ enhance(Ship): void
+ getCollider(): Collider
+ getShape(): Shape
+ getState(): PowerUpState
+ hit(int): void
+ update(): void
DoubleDamagePowerUp
+ enhance(Ship): void
DoubleShotPowerUp
+ enhance(Ship): void
SpreadPowerUp
+ enhance(Ship): void
Laser
+ removeAttachment(): Gun
+ shoot(Vector2D, float, float): Bullet [ ]
«interface»
Gun
+ removeAttachment(): Gun
+ shoot(Vector2D, float, float): Bullet [ ]
DoubleShot
- gun: Gun
+ removeAttachment(): Gun
+ shoot(Vector2D, float, float): Bullet [ ]
Spread
- gun: Gun
+ removeAttachment(): Gun
+ shoot(Vector2D, float, float): Bullet [ ]
DoubleDamage
- gun: Gun
+ removeAttachment(): Gun
+ shoot(Vector2D, float, float): Bullet [ ]
«interface»
Vector2D
+ add(Vector2D): Vector2D
+ angle(): float
+ getX(): float
+ getY(): float
+ module(): float
+ multiply(float): Vector2D
+ rotate(float): Vector2D
+ subtract(Vector2D): Vector2D
+ unitary(): Vector2D
Observable<BulletObserver>
Observable<ShipObserver>
«interface»
Observable<T>
+ addObserver(T): void
+ removeObserver(T): void
«interface»
ShipObserver
+ notifyDestroy(): void
+ notifyHit(int): void
+ notifyShoot(Bullet [ ]): void
«interface»
BulletObserver
+ notifyDestroy(GameEntity): void
+ notifyHit(GameEntity): void
Observable<SpawnObserver<Bullet>>
ShootingObserver
- observers: SpawnObserver<Bullet> [0..*]
+ addObserver(SpawnObserver<Bullet>): void
+ notifyDestroy(): void
+ notifyHit(int): void
+ notifyShoot(Bullet [ ]): void
+ removeObserver(SpawnObserver<Bullet>): void
PointScout
- destroyVisitor: EntityVisitor<int>
- hitVisitor: EntityVisitor<int>
- player: Player
+ notifyDestroy(GameEntity): void
+ notifyHit(GameEntity): void
StarShipPlayer
- hp: int
- initialHp: int
- lives: int
- name: String
- number: int
- points: int
+ addPoints(int): void
+ getHp(): int
+ getLifes(): int
+ getLives(): int
+ getName(): String
+ getNumber(): int
+ getPoints(): int
+ isAlive(): boolean
«interface»
Player
+ addPoints(int): void
+ getHp(): int
+ getLives(): int
+ getName(): String
+ getNumber(): int
+ getPoints(): int
+ isAlive(): boolean
HitVisitor
+ visitAsteroid(Asteroid): int
+ visitBullet(Bullet): int
+ visitPowerUp(PowerUp): int
+ visitShip(SmallShip): int
"EntityVisitor<int>"
DestroyVisitor
+ visitAsteroid(Asteroid): int
+ visitBullet(Bullet): int
+ visitPowerUp(PowerUp): int
+ visitShip(SmallShip): int
"EntityVisitor<int>"


«interface»
Observable<T>
+ addObserver(T): void
+ removeObserver(T): void
Observable<SpawnObserver<T>>
«interface»
ShipObserver
+ notifyDestroy(): void
+ notifyHit(int): void
+ notifyShoot(Bullet [ ]): void
«interface»
SpawnObserver<T>
+ notifySpawn(T): void
«interface»
Spawner<T>
+ addObserver(SpawnObserver<T>): void
+ removeObserver(SpawnObserver<T>): void
+ spawn(int, int): T
Spawner<Asteroid>
AsteroidSpawner
- observers: SpawnObserver<Asteroid> [0..*]
+ addObserver(SpawnObserver<Asteroid>): void
+ removeObserver(SpawnObserver<Asteroid>): void
+ spawn(int, int): Asteroid
Spawner<Ship>
«interface»
ShipSpawner
+ addObserver(SpawnObserver<T>): void
+ removeObserver(SpawnObserver<T>): void
+ respawn(Ship, int, int): Ship
+ spawn(int, int): T
ShipSpawner
- observers: SpawnObserver<Ship> [0..*]
+ addObserver(SpawnObserver<Ship>): void
+ removeObserver(SpawnObserver<Ship>): void
+ respawn(Ship, int, int): void
+ spawn(int, int): Ship
Spawner<PowerUp>
PowerUpSpawner
- observers: SpawnObserver<PowerUp> [0..*]
+ addObserver(SpawnObserver<PowerUp>): void
+ removeObserver(SpawnObserver<PowerUp>): void
+ spawn(int, int): PowerUp
«interface»
SpawnPainter<T>
+ getNew(): Drawable [ ]
+ notifySpawn(T): void
BaseSpawnPainter<T>
- buffer: Drawable [0..*]
- paintor: Painter<T>
+ getNew(): Drawable [ ]
+ notifySpawn(T): void
«interface»
Drawable
+ draw(Plane): void
+ isActive(): boolean
«interface»
Painter<T>
+ draw(T, Color): Drawable
ShipPainter
+ draw(Ship, Color): Drawable
Painter<Ship>
AsteroidPainter
+ draw(Asteroid, Color): Drawable
Painter<Asteroid>
PowerUpPainter
+ draw(PowerUp, Color): Drawable
Painter<PowerUp>
BulletPainter
+ draw(Bullet, Color): Drawable
Painter<Bullet>
ShipState
+ destroyed: boolean
+ heading: float
+ hp: int
- invincible: boolean
+ position: Vector2D
+ shooting: boolean
+ speed: float
ShipView
- color: Color
- shapePoints: Vector2D [0..*]
- ship: ShipState
+ draw(Plane): void
+ isActive(): boolean
AsteroidState
+ destroyed: boolean
+ heading: float
+ hp: int
+ position: Vector2D
+ speed: float
AsteroidView
- asteroid: AsteroidState
- color: Color
- shapePoints: Vector2D [0..*]
+ draw(Plane): void
+ isActive(): boolean
BulletState
+ destroyed: boolean
+ heading: float
+ position: Vector2D
+ speed: float
BulletView
- bullet: BulletState
- shapePoints: Vector2D [0..*]
+ draw(Plane): void
+ isActive(): boolean
«interface»
Player
+ addPoints(int): void
+ getHp(): int
+ getLives(): int
+ getName(): String
+ getNumber(): int
+ getPoints(): int
+ isAlive(): boolean
PlayerView
- fontSize: int
- player: Player
- position: Vector2D
+ draw(Plane): void
+ isActive(): boolean
SpreadView
- color: Color
- shapePoints: Vector2D
- state: PowerUpState
+ draw(Plane): void
+ isActive(): boolean
DoubleShotView
- color: Color
- shapePoints: Vector2D [0..*]
- state: PowerUpState
+ draw(Plane): void
+ isActive(): boolean
PowerUpState
+ destroyed: boolean
+ heading: float
+ position: Vector2D
+ timeLeft: int
DoubleDamageView
- color: Color
- shapePoints: Vector2D [0..*]
- state: PowerUpState
+ draw(Plane): void
+ isActive(): boolean
BlinkingPowerUpView
- powerUpState: PowerUpState
- powerUpView: Drawable
+ draw(Plane): void
+ isActive(): boolean