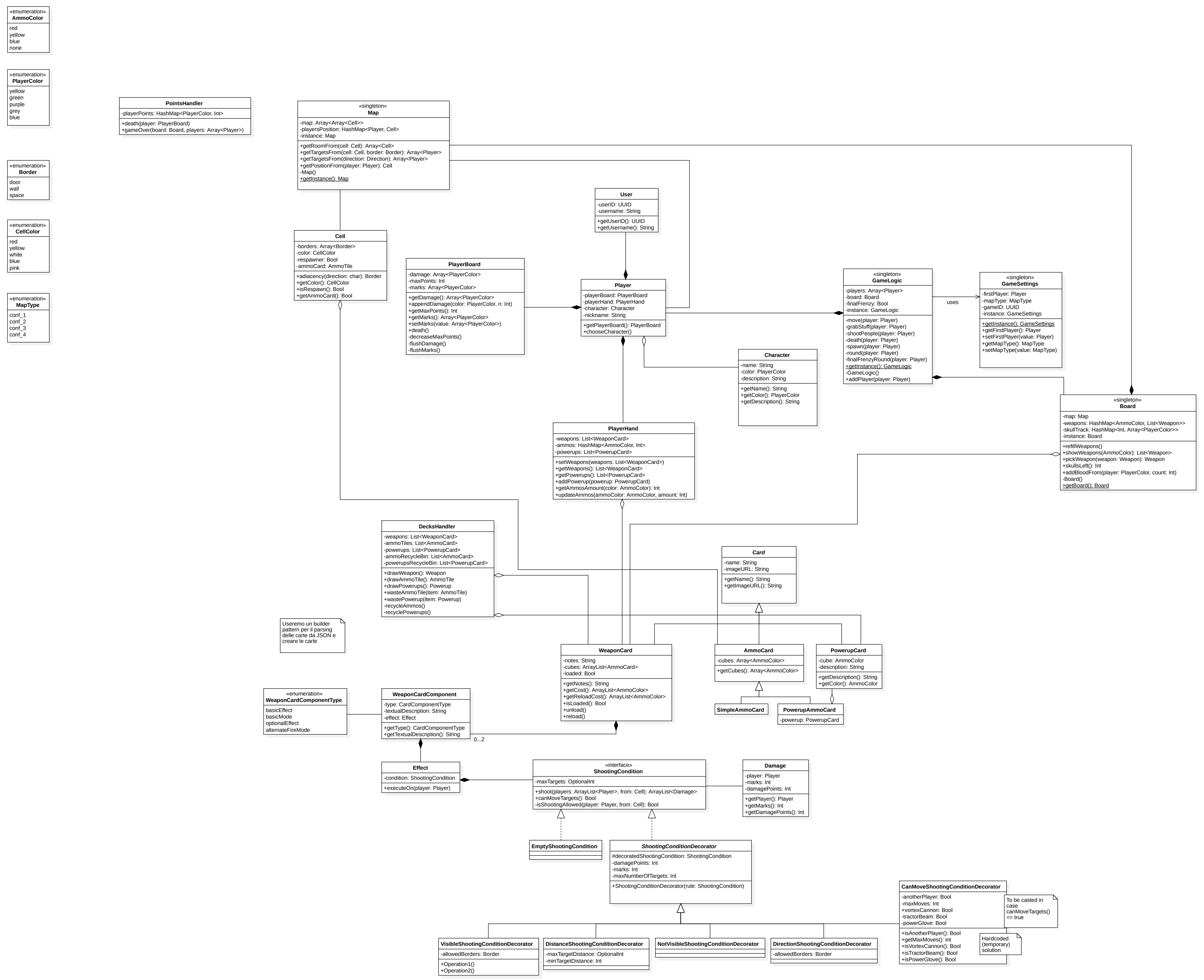
Model::Main


«enumeration»
AmmoColor
red
yellow
blue
none
«enumeration»
PlayerColor
yellow
green
purple
grey
blue
«enumeration»
Border
door
wall
space
«enumeration»
CellColor
red
yellow
white
blue
pink
«enumeration»
MapType
conf_1
conf_2
conf_3
conf_4
PointsHandler
-playerPoints: HashMap<PlayerColor, Int>
+death(player: PlayerBoard)
+gameOver(board: Board, players: Array<Player>)
«singleton»
Map
-map: Array<Array<Cell>>
-playersPosition: HashMap<Player, Cell>
-instance: Map
+getRoomFrom(cell: Cell): Array<Cell>
+getTargetsFrom(cell: Cell, border: Border): Array<Player>
+getTargetsFrom(direction: Direction): Array<Player>
+getPositionFrom(player: Player): Cell
-Map()
+getInstance(): Map
Cell
-borders: Array<Border>
-color: CellColor
-respawner: Bool
-ammoCard: AmmoTile
+adiacency(direction: char): Border
+getColor(): CellColor
+isRespawn(): Bool
+getAmmoCard(): Bool
User
-userID: UUID
-username: String
+getUserID(): UUID
+getUsername(): String
PlayerBoard
-damage: Array<PlayerColor>
-maxPoints: Int
-marks: Array<PlayerColor>
+getDamage(): Array<PlayerColor>
+appendDamage(color: PlayerColor, n: Int)
+getMaxPoints(): Int
+getMarks(): Array<PlayerColor>
+setMarks(value: Array<PlayerColor>)
+death()
-decreaseMaxPoints()
-flushDamage()
-flushMarks()
Player
-playerBoard: PlayerBoard
-playerHand: PlayerHand
-character: Character
-nickname: String
+getPlayerBoard(): PlayerBoard
+chooseCharacter()
«singleton»
GameLogic
-players: Array<Player>
-board: Board
-finalFrenzy: Bool
-instance: GameLogic
-move(player: Player)
-grabStuff(player: Player)
-shootPeople(player: Player)
-death(player: Player)
-spawn(player: Player)
-round(player: Player)
-finalFrenzyRound(player: Player)
+getInstance(): GameLogic
-GameLogic()
+addPlayer(player: Player)
uses
«singleton»
GameSettings
-firstPlayer: Player
-mapType: MapType
-gameID: UUID
-instance: GameSettings
+getInstance(): GameSettings
+getFirstPlayer(): Player
+setFirstPlayer(value: Player)
+getMapType(): MapType
+setMapType(value: MapType)
Character
-name: String
-color: PlayerColor
-description: String
+getName(): String
+getColor(): PlayerColor
+getDescription(): String
«singleton»
Board
-map: Map
-weapons: HashMap<AmmoColor, List<Weapon>>
-skullTrack: HashMap<Int, Array<PlayerColor>>
-instance: Board
+refillWeapons()
+showWeapons(AmmoColor): List<Weapon>
+pickWeapon(weapon: Weapon): Weapon
+skullsLeft(): Int
+addBloodFrom(player: PlayerColor, count: Int)
-Board()
+getBoard(): Board
PlayerHand
-weapons: List<WeaponCard>
-ammos: HashMap<AmmoColor, Int>
-powerups: List<PowerupCard>
+setWeapons(weapons: List<WeaponCard>)
+getWeapons(): List<WeaponCard>
+getPowerups(): List<PowerupCard>
+addPowerup(powerup: PowerupCard)
+getAmmosAmount(color: AmmoColor): Int
+updateAmmos(ammoColor: AmmoColor, amount: Int)
DecksHandler
-weapons: List<WeaponCard>
-ammoTiles: List<AmmoCard>
-powerups: List<PowerupCard>
-ammoRecycleBin: List<AmmoCard>
-powerupsRecycleBin: List<PowerupCard>
+drawWeapon(): Weapon
+drawAmmoTile(): AmmoTile
+drawPowerups(): Powerup
+wasteAmmoTile(item: AmmoTile)
+wastePowerup(item: Powerup)
-recycleAmmos()
-recyclePowerups()
Card
-name: String
-imageURL: String
+getName(): String
+getImageURL(): String
Useremo un builder pattern per il parsing delle carte da JSON e creare le carte
WeaponCard
-notes: String
-cubes: ArrayList<AmmoCard>
-loaded: Bool
+getNotes(): String
+getCost(): ArrayList<AmmoColor>
+getReloadCost(): ArrayList<AmmoColor>
+isLoaded(): Bool
+unload()
+reload()
AmmoCard
-cubes: Array<AmmoColor>
+getCubes(): Array<AmmoColor>
PowerupCard
-cube: AmmoColor
-description: String
+getDescription(): String
+getColor(): AmmoColor
SimpleAmmoCard
PowerupAmmoCard
-powerup: PowerupCard
«enumeration»
WeaponCardComponentType
basicEffect
basicMode
optionalEffect
alternateFireMode
WeaponCardComponent
-type: CardComponentType
-textualDescription: String
-effect: Effect
+getType(): CardComponentType
+getTextualDescription(): String
0...2
Effect
-condition: ShootingCondition
+executeOn(player: Player)
«interface»
ShootingCondition
-maxTargets: OptionalInt
+shoot(players: ArrayList<Player>, from: Cell): ArrayList<Damage>
+canMoveTargets(): Bool
-isShootingAllowed(player: Player, from: Cell): Bool
Damage
-player: Player
-marks: Int
-damagePoints: Int
+getPlayer(): Player
+getMarks(): Int
+getDamagePoints(): Int
EmptyShootingCondition
ShootingConditionDecorator
#decoratedShootingCondition: ShootingCondition
-damagePoints: Int
-marks: Int
-maxNumberOfTargets: Int
+ShootingConditionDecorator(rule: ShootingCondition)
CanMoveShootingConditionDecorator
-anotherPlayer: Bool
-maxMoves: Int
+vortexCannon: Bool
-tractorBeam: Bool
-powerGlove: Bool
+isAnotherPlayer(): Bool
+getMaxMoves(): int
+isVortexCannon(): Bool
+isTractorBeam(): Bool
+isPowerGlove(): Bool
To be casted in case canMoveTargets() == true
Hardcoded (temporary) solution
VisibleShootingConditionDecorator
-allowedBorders: Border
+Operation1()
+Operation2()
DistanceShootingConditionDecorator
-maxTargetDistance: OptionalInt
-minTargetDistance: Int
NotVisibleShootingConditionDecorator
DirectionShootingConditionDecorator
-allowedBorders: Border