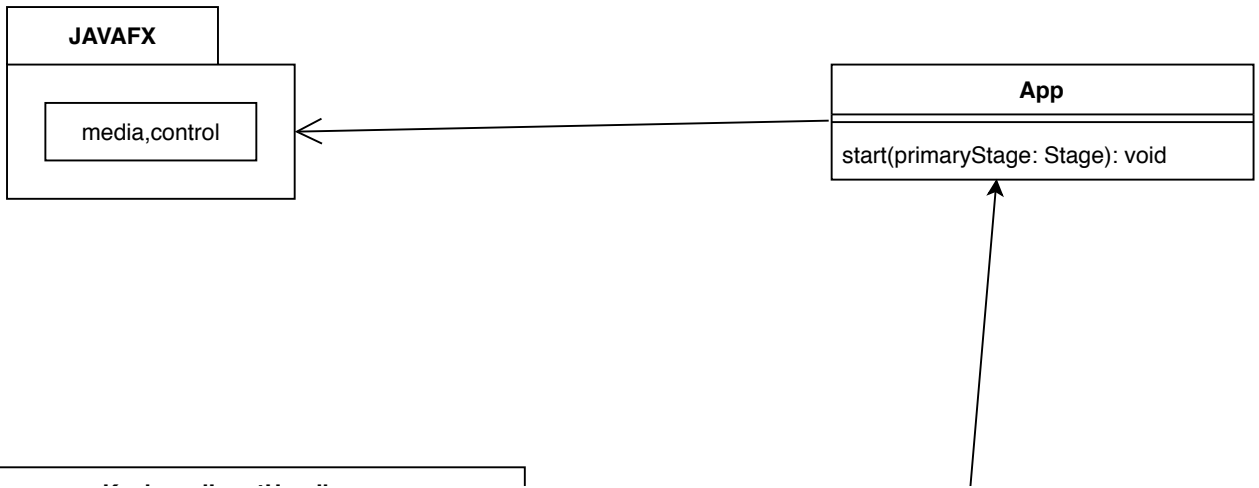
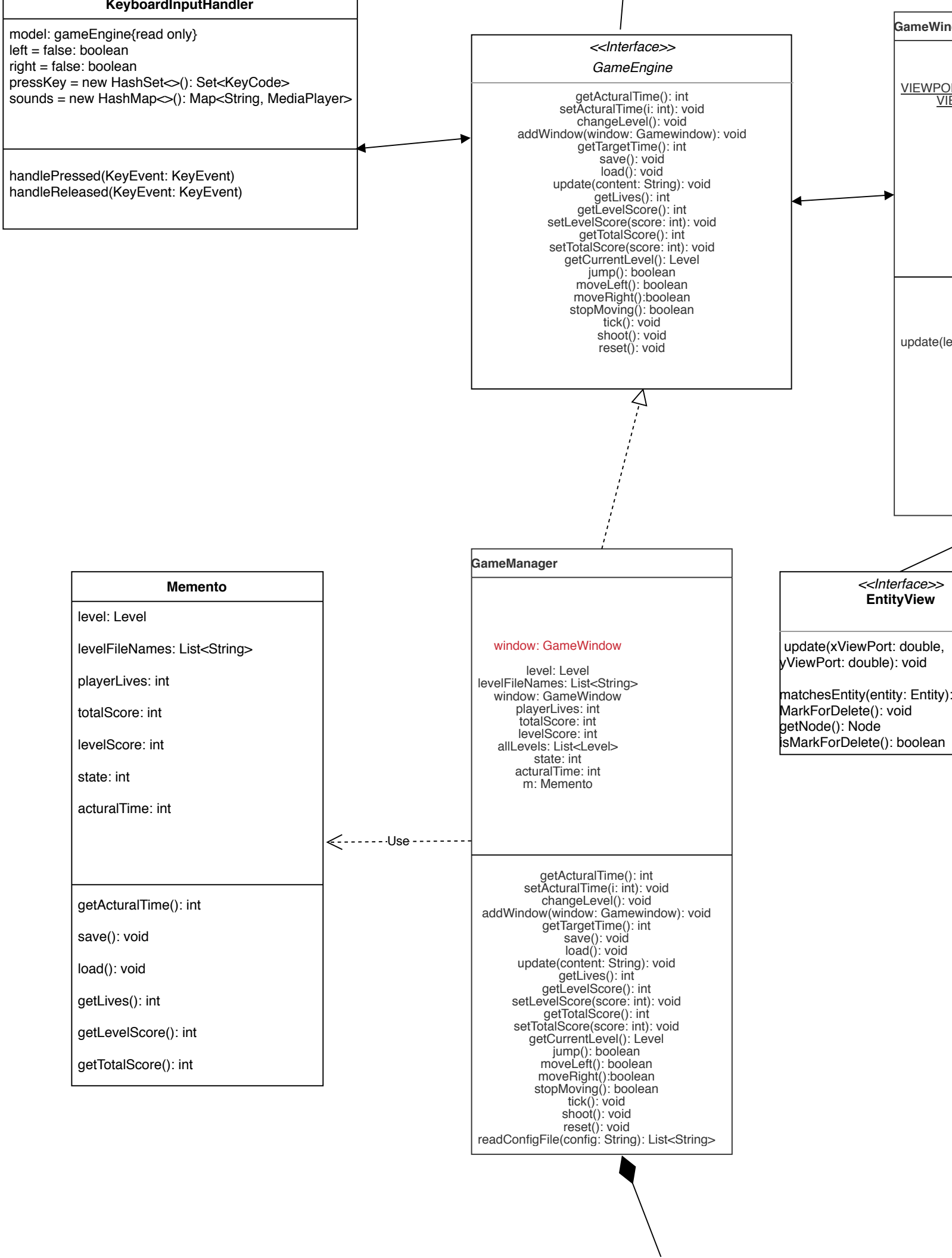
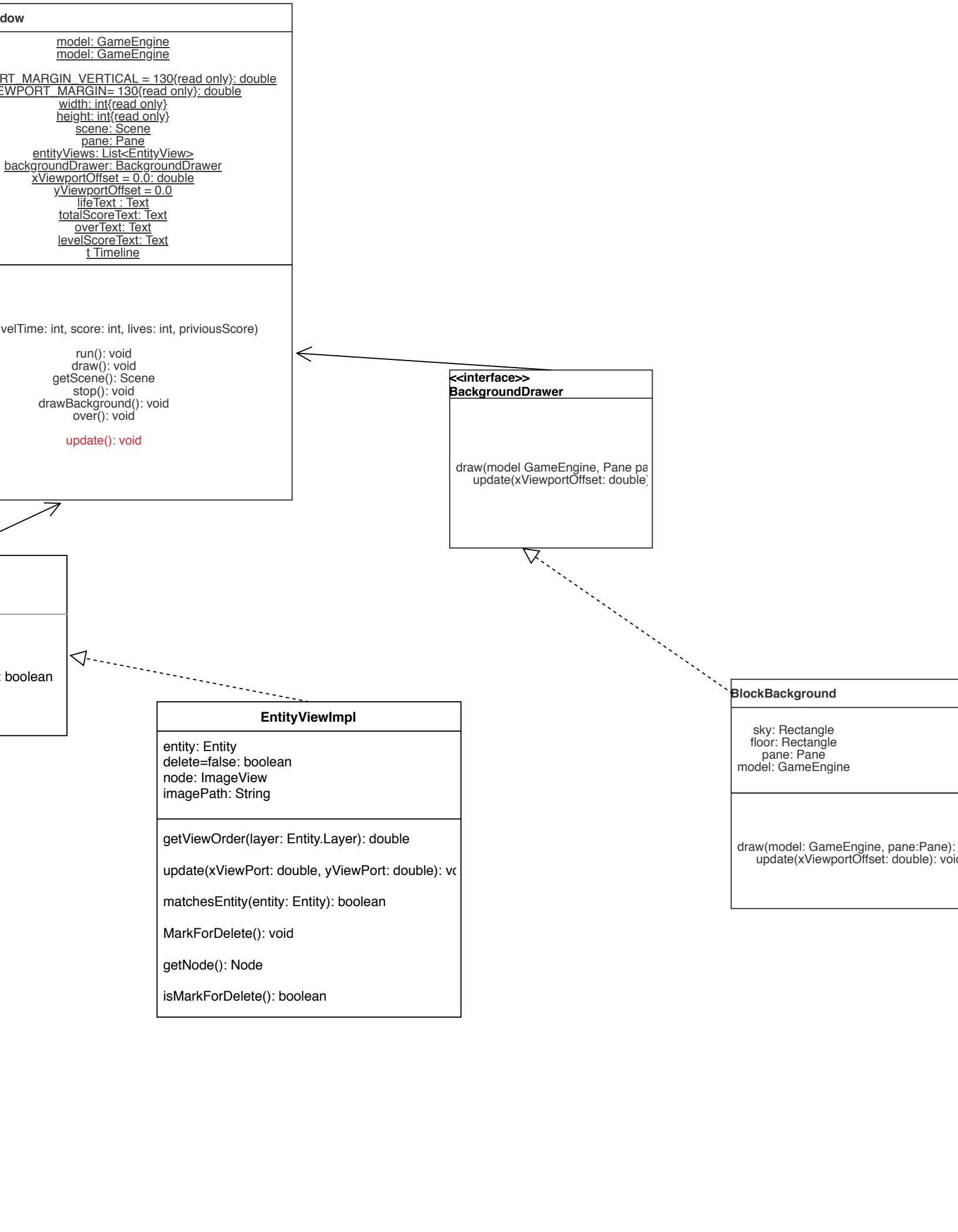
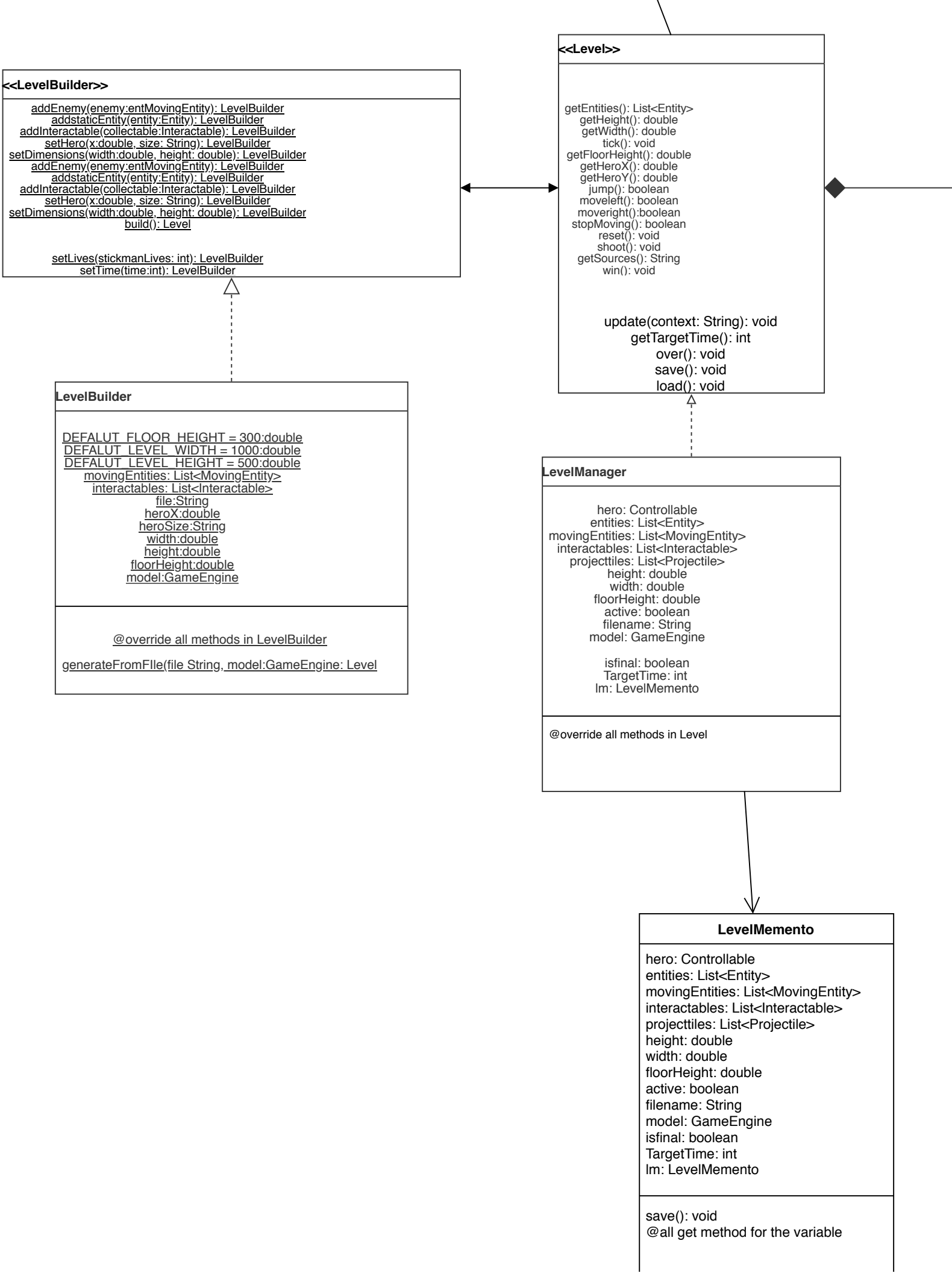
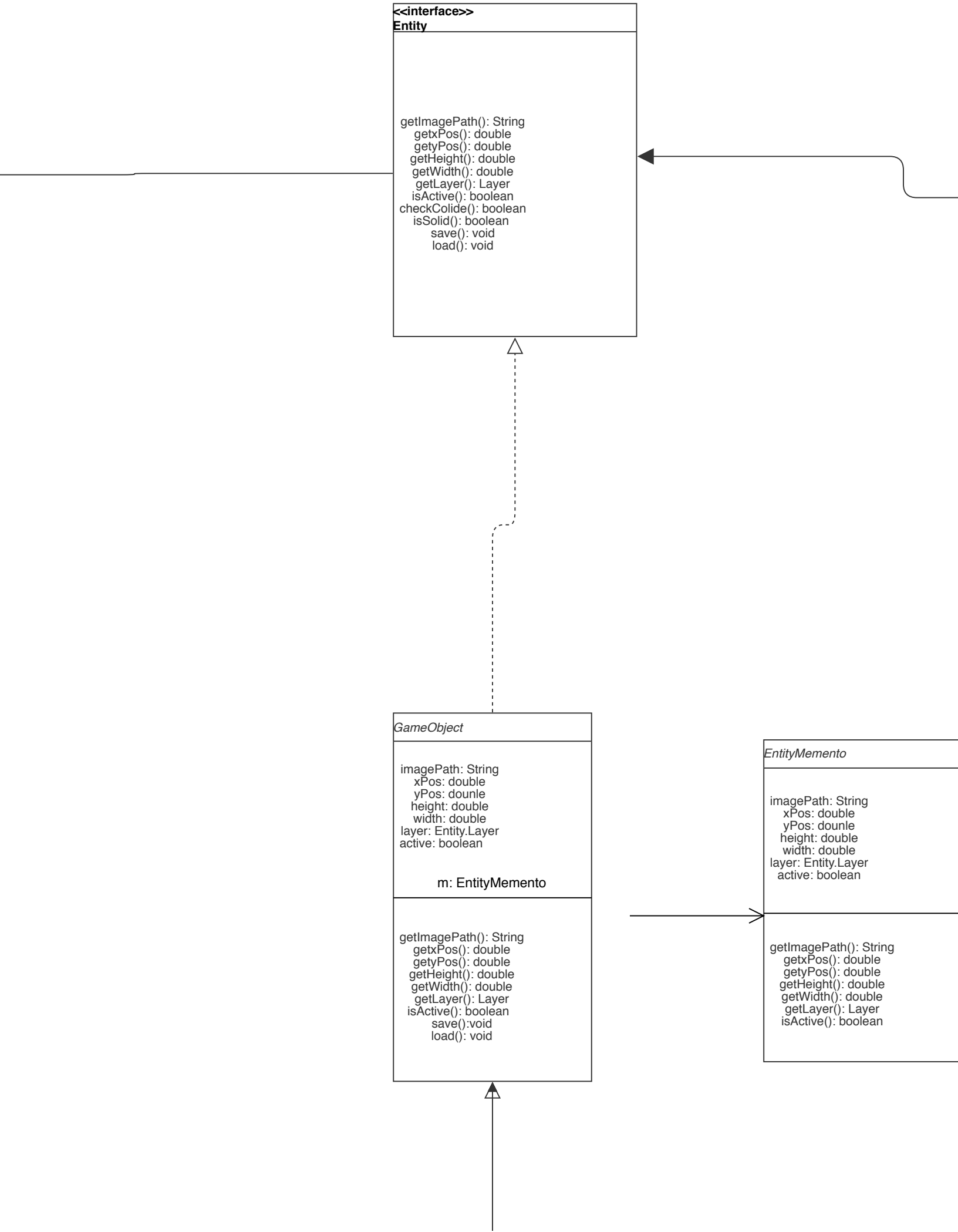

JAVAFX
media,control
App
start(primaryStage: Stage): void

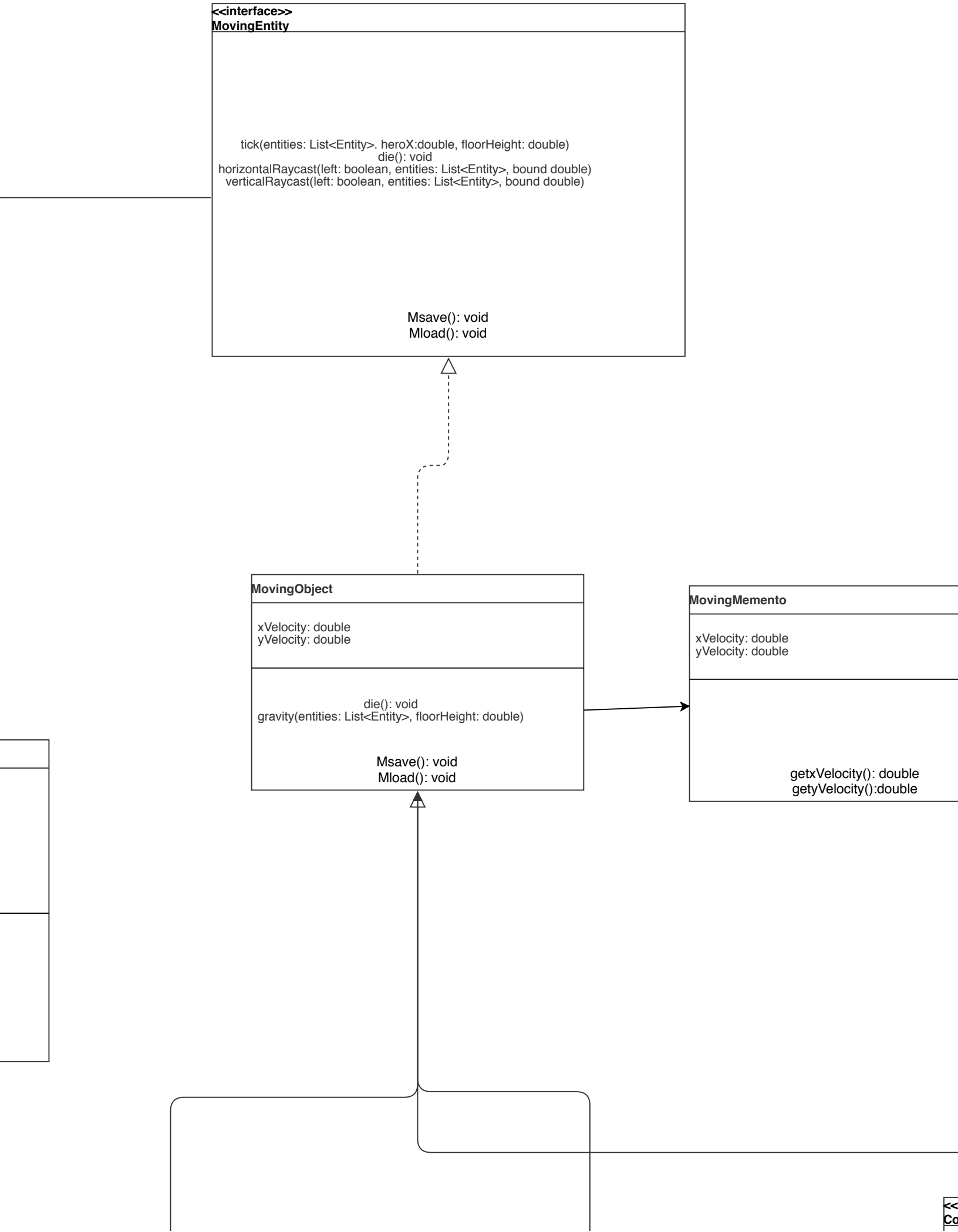

KeyboardInputHandler
model: gameEngine{read only}
left = false: boolean
right = false: boolean
pressKey = new HashSet<>(): Set<KeyCode>
sounds = new HashMap<>(): Map<String, MediaPlayer>
handlePressed(KeyEvent: KeyEvent)
handleReleased(KeyEvent: KeyEvent)
<<Interface>>
GameEngine
getActuralTime(): int
setActuralTime(i: int): void
changeLevel(): void
addWindow(window: Gamewindow): void
getTargetTime(): int
save(): void
load(): void
update(content: String): void
getLives(): int
getLevelScore(): int
setLevelScore(score: int): void
getTotalScore(): int
setTotalScore(score: int): void
getCurrentLevel(): Level
jump(): boolean
moveLeft(): boolean
moveRight():boolean
stopMoving(): boolean
tick(): void
shoot(): void
reset(): void
GameWin
VIEWPO
VI
update(le
Memento
level: Level
levelFileNames: List<String>
playerLives: int
totalScore: int
levelScore: int
state: int
acturalTime: int
getActuralTime(): int
save(): void
load(): void
getLives(): int
getLevelScore(): int
getTotalScore(): int
Use
GameManager
window: GameWindow
level: Level
levelFileNames: List<String>
window: GameWindow
playerLives: int
totalScore: int
levelScore: int
allLevels: List<Level>
state: int
acturalTime: int
m: Memento
getActuralTime(): int
setActuralTime(i: int): void
changeLevel(): void
addWindow(window: Gamewindow): void
getTargetTime(): int
save(): void
load(): void
update(content: String): void
getLives(): int
getLevelScore(): int
setLevelScore(score: int): void
getTotalScore(): int
setTotalScore(score: int): void
getCurrentLevel(): Level
jump(): boolean
moveLeft(): boolean
moveRight():boolean
stopMoving(): boolean
tick(): void
shoot(): void
reset(): void
readConfigFile(config: String): List<String>
<<Interface>>
EntityView
update(xViewPort: double,
yViewPort: double): void
matchesEntity(entity: Entity):
MarkForDelete(): void
getNode(): Node
isMarkForDelete(): boolean

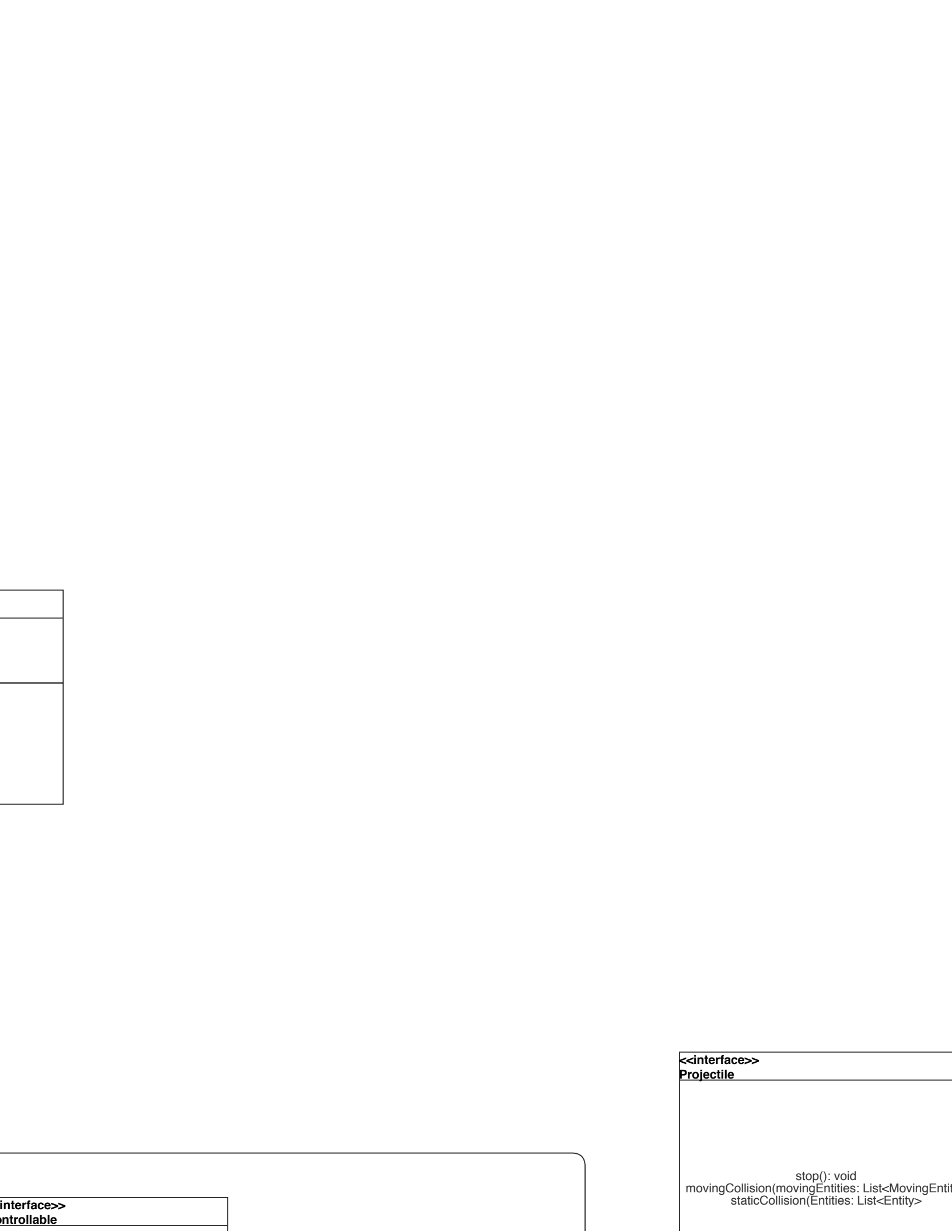

dow
model: GameEngine
model: GameEngine
RT_MARGIN_VERTICAL = 130{read only}: double
EWPORT_MARGIN= 130{read only}: double
width: int{read only}
height: int{read only}
scene: Scene
pane: Pane
entityViews: List<EntityView>
backgroundDrawer: BackgroundDrawer
xViewportOffset = 0.0: double
yViewportOffset = 0.0
lifeText : Text
totalScoreText: Text
overText: Text
levelScoreText: Text
t Timeline
velTime: int, score: int, lives: int, priviousScore)
run(): void
draw(): void
getScene(): Scene
stop(): void
drawBackground(): void
over(): void
update(): void
<<interface>>
BackgroundDrawer
draw(model GameEngine, Pane pa
update(xViewportOffset: double)
: boolean
EntityViewImpl
entity: Entity
delete=false: boolean
node: ImageView
imagePath: String
getViewOrder(layer: Entity.Layer): double
update(xViewPort: double, yViewPort: double): vo
matchesEntity(entity: Entity): boolean
MarkForDelete(): void
getNode(): Node
isMarkForDelete(): boolean
BlockBackground
sky: Rectangle
floor: Rectangle
pane: Pane
model: GameEngine
draw(model: GameEngine, pane:Pane):
update(xViewportOffset: double): voi

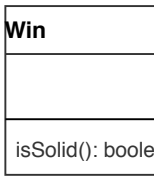
vc
d

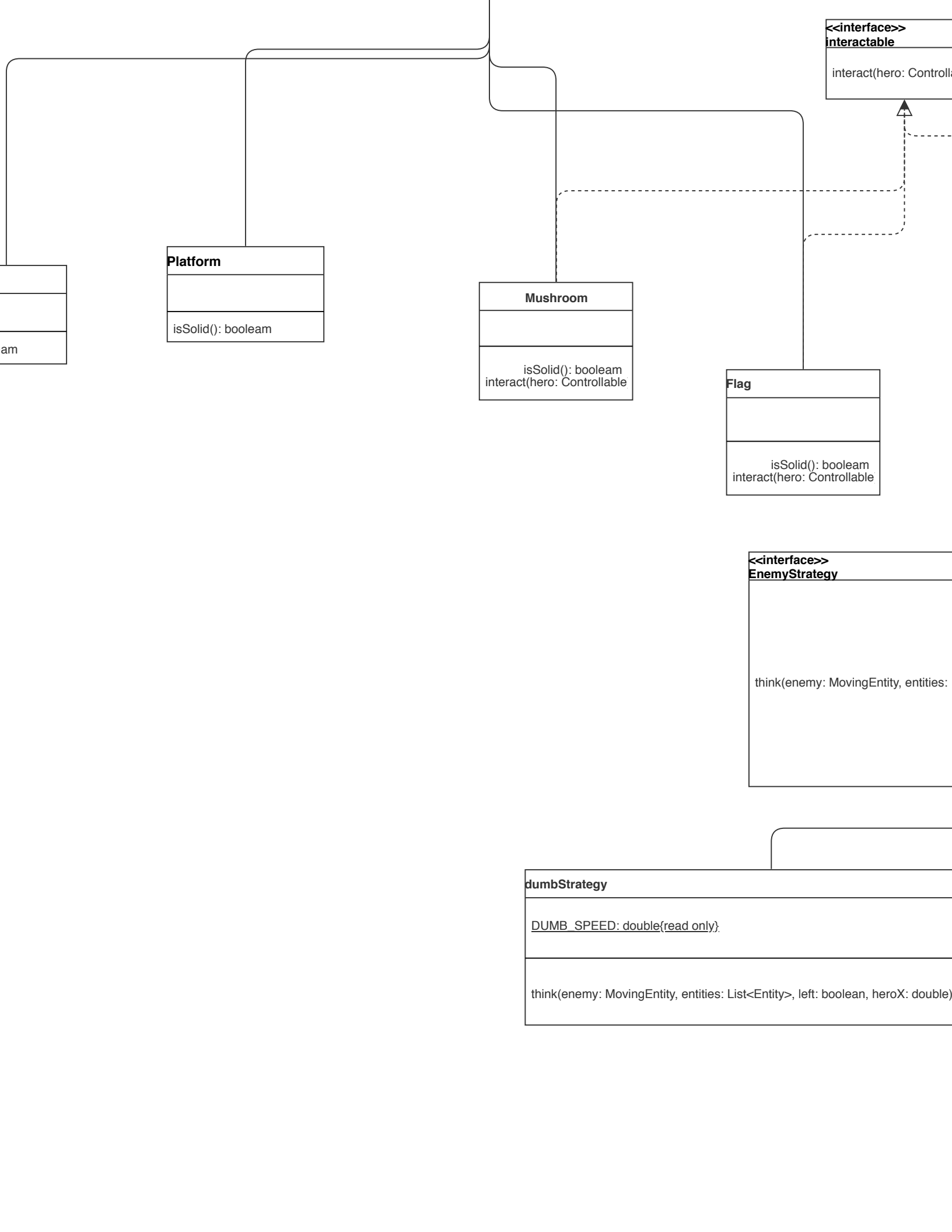

<<Level>>
getEntities(): List<Entity>
getHeight(): double
getWidth(): double
tick(): void
getFloorHeight(): double
getHeroX(): double
getHeroY(): double
jump(): boolean
moveleft(): boolean
moveright():boolean
stopMoving(): boolean
reset(): void
shoot(): void
getSources(): String
win(): void
update(context: String): void
getTargetTime(): int
over(): void
save(): void
load(): void
<<LevelBuilder>>
addEnemy(enemy:entMovingEntity): LevelBuilder
addstaticEntity(entity:Entity): LevelBuilder
addInteractable(collectable:Interactable): LevelBuilder
setHero(x:double, size: String): LevelBuilder
setDimensions(width:double, height: double): LevelBuilder
addEnemy(enemy:entMovingEntity): LevelBuilder
addstaticEntity(entity:Entity): LevelBuilder
addInteractable(collectable:Interactable): LevelBuilder
setHero(x:double, size: String): LevelBuilder
setDimensions(width:double, height: double): LevelBuilder
build(): Level
setLives(stickmanLives: int): LevelBuilder
setTime(time:int): LevelBuilder
LevelBuilder
DEFALUT_FLOOR_HEIGHT = 300:double
DEFALUT_LEVEL_WIDTH = 1000:double
DEFALUT_LEVEL_HEIGHT = 500:double
movingEntities: List<MovingEntity>
interactables: List<Interactable>
file:String
heroX:double
heroSize:String
width:double
height:double
floorHeight:double
model:GameEngine
@override all methods in LevelBuilder
generateFromFIle(file String, model:GameEngine: Level
LevelManager
hero: Controllable
entities: List<Entity>
movingEntities: List<MovingEntity>
interactables: List<Interactable>
projecttiles: List<Projectile>
height: double
width: double
floorHeight: double
active: boolean
filename: String
model: GameEngine
isfinal: boolean
TargetTime: int
lm: LevelMemento
@override all methods in Level
LevelMemento
hero: Controllable
entities: List<Entity>
movingEntities: List<MovingEntity>
interactables: List<Interactable>
projecttiles: List<Projectile>
height: double
width: double
floorHeight: double
active: boolean
filename: String
model: GameEngine
isfinal: boolean
TargetTime: int
lm: LevelMemento
save(): void
@all get method for the variable

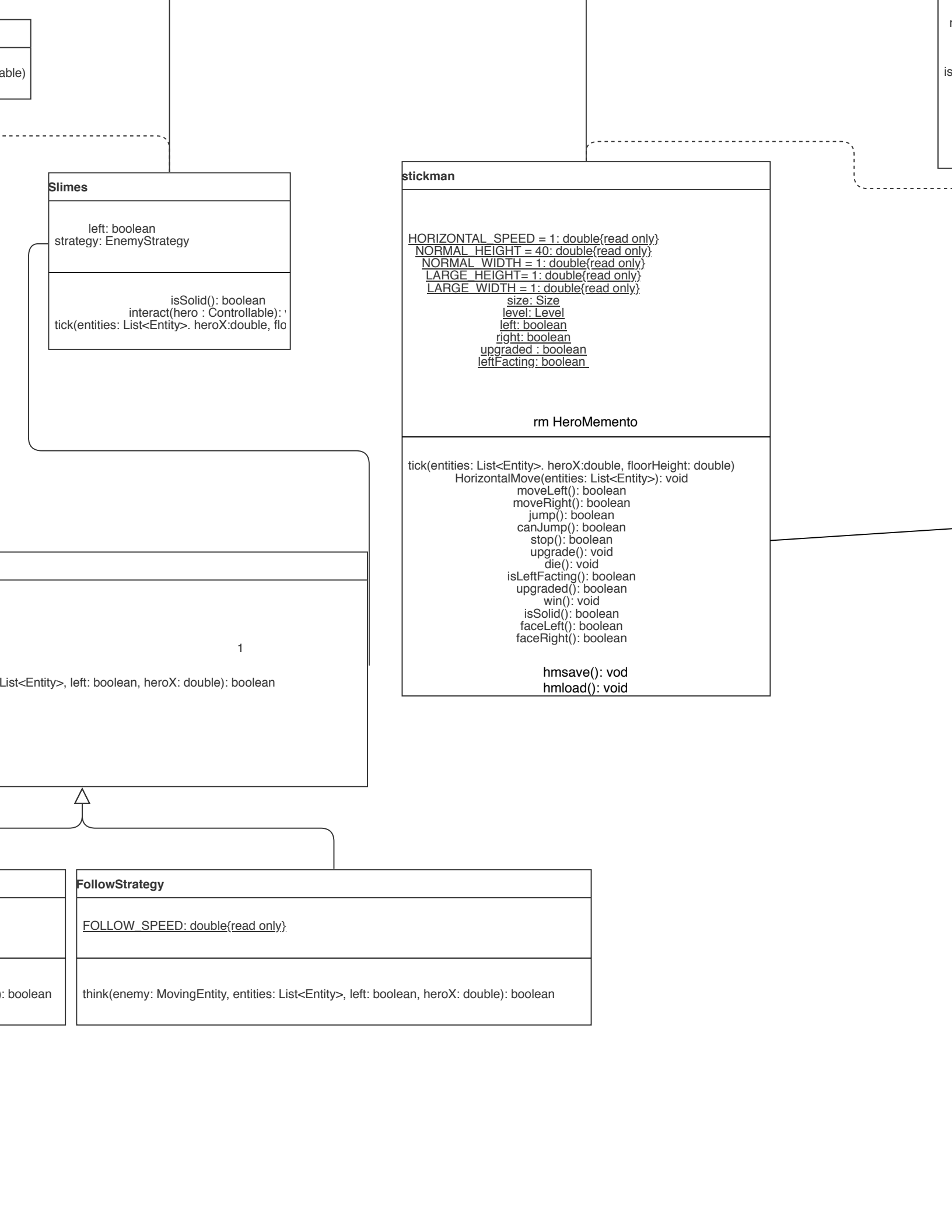

<<interface>>
Entity
getImagePath(): String
getxPos(): double
getyPos(): double
getHeight(): double
getWidth(): double
getLayer(): Layer
isActive(): boolean
checkColide(): boolean
isSolid(): boolean
save(): void
load(): void
GameObject
imagePath: String
xPos: double
yPos: dounle
height: double
width: double
layer: Entity.Layer
active: boolean
m: EntityMemento
getImagePath(): String
getxPos(): double
getyPos(): double
getHeight(): double
getWidth(): double
getLayer(): Layer
isActive(): boolean
save():void
load(): void
EntityMemento
imagePath: String
xPos: double
yPos: dounle
height: double
width: double
layer: Entity.Layer
active: boolean
getImagePath(): String
getxPos(): double
getyPos(): double
getHeight(): double
getWidth(): double
getLayer(): Layer
isActive(): boolean


<<interface>>
MovingEntity
tick(entities: List<Entity>. heroX:double, floorHeight: double)
die(): void
horizontalRaycast(left: boolean, entities: List<Entity>, bound double)
verticalRaycast(left: boolean, entities: List<Entity>, bound double)
Msave(): void
Mload(): void
MovingObject
xVelocity: double
yVelocity: double
die(): void
gravity(entities: List<Entity>, floorHeight: double)
Msave(): void
Mload(): void
MovingMemento
xVelocity: double
yVelocity: double
getxVelocity(): double
getyVelocity():double


interface>>
ntrollable
<<interface>>
Projectile
stop(): void
movingCollision(movingEntities: List<MovingEntit
staticCollision(Entities: List<Entity>


Win
isSolid(): boole


am
Platform
isSolid(): booleam
Mushroom
isSolid(): booleam
interact(hero: Controllable
Flag
isSolid(): booleam
interact(hero: Controllable
<<interface>>
interactable
interact(hero: Controlla
<<interface>>
EnemyStrategy
think(enemy: MovingEntity, entities:
dumbStrategy
DUMB_SPEED: double{read only}
think(enemy: MovingEntity, entities: List<Entity>, left: boolean, heroX: double)


able)
Slimes
left: boolean
strategy: EnemyStrategy
isSolid(): boolean
interact(hero : Controllable):
tick(entities: List<Entity>. heroX:double, flo
stickman
HORIZONTAL_SPEED = 1: double{read only}
NORMAL_HEIGHT = 40: double{read only}
NORMAL_WIDTH = 1: double{read only}
LARGE_HEIGHT= 1: double{read only}
LARGE_WIDTH = 1: double{read only}
size: Size
level: Level
left: boolean
right: boolean
upgraded : boolean
leftFacting: boolean
rm HeroMemento
tick(entities: List<Entity>. heroX:double, floorHeight: double)
HorizontalMove(entities: List<Entity>): void
moveLeft(): boolean
moveRight(): boolean
jump(): boolean
canJump(): boolean
stop(): boolean
upgrade(): void
die(): void
isLeftFacting(): boolean
upgraded(): boolean
win(): void
isSolid(): boolean
faceLeft(): boolean
faceRight(): boolean
hmsave(): vod
hmload(): void
1
List<Entity>, left: boolean, heroX: double): boolean
FollowStrategy
FOLLOW_SPEED: double{read only}
think(enemy: MovingEntity, entities: List<Entity>, left: boolean, heroX: double): boolean
: boolean

moveLeft(): boolean
moveRight(): boolean
jump(): boolean
stop(): boolean
upgrade(): void
sLeftFacting(): boolean
upgraded(): boolean
win(): void

hmsave(): void
hmload(): void

Bullet

BULLET_SPEED = 2: double{read only}
BULLET_HEIGHT = 10: double{read only}
BULLET_WIDTH = 10: double{read only}

tick(entities: List<Entity>. heroX:double, floorHeight: double)
stop(): void

HeroMemento

level: Level
left: boolean
right: boolean
upgraded : boolean
leftFacting: boolean

getLevel(): Level
getLeft():boolean
getRight(): boolean
isUpgraded(); boolean
isLeftFacing(): boolean