System design document for the Forest Frenzy project (SDD)

Contents

Version: *Forest Frenzy, last version.*

Date: 2013-05-26

Author: Group 18

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The game must be designed in such a way that it will be easy to modify. It should be possible to switch GUI and isolate parts of the application for testing. We will aim to make the components of the application as stand alone as possible.

Though, we will not create a graphical menu for this modification. What we mean is that our graphical components will be as stand alone as possible.

1.2 Definitions, acronyms and abbreviations

- **GUI**, Graphical User Interface.
- **Java**, platform independent programming language.
- **JRE**, the Java run time Environment. Additional software needed to be downloaded.
- **Eclipse**, Integrated development environment for Java.
- **Slick2D**, external library for game development.
- **JBox2D**, external physics engine for java development.
- **MVC**, a way to organize and partition an application with a GUI into three distinct parts, model, view, controller.

2 System design

2.1 Overview

The application will use a modified MVC model.

2.1.1 The model functionality

The models functionality will be exposed by the interface IEntityModel. In addition to this there is another interface IAliveModel and ICollectibleModel. IEntityModel will be the top level interface and there other interfaces will extend it, see Figure 1. All living entities in the game world will implement IAliveModel, all collectible entities will implement ICollectibleModel while every other entity will in some way implement IEntityModel. In

addition to these interfaces there will be a few interfaces and a few abstract classes that expose the functionality of the affected models even further.

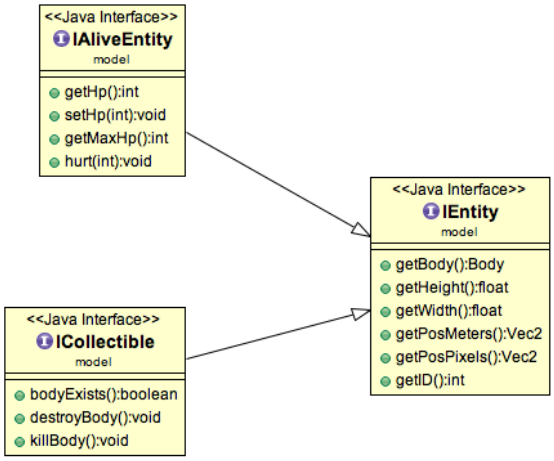


Figure 1: Model and functionality (interfaces)

2.1.2 The controller functinality

The controller functionality will be exposed by the interface IEntityController, see Figure 2. This one interface will not be split into smaller, more specific interfaces since there is no risk it will become too large and diverse.

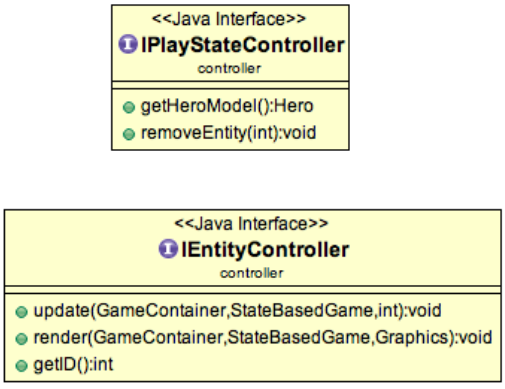


Figure 2: Controller and functionality (interface)

2.1.3 Unique identifiers

Every entity, or model, in the game world will be given a unique identification number so that it can be removed easily together with it's controller from the game world when needed. Every other entity of the application will not be given any unique identifiers.

2.1.4 Event handling

Slick2D has a built in event handling system that will let the application react to inputs from the user. The application will implemented in a way that every entity will handle their own events so that if changes need to be applied, they'll be localized.
Entities in the game world will not handle any collision detection though, this will be handled by a separate class using JBox2D's interface ContactListener.

2.1.5 Sounds and music

Sound and music will be implemented in the application and to let them be easily modified we decided to develop an individual class Sounds with support from Slick2D that only handles sounds and music, see Figure 3.

2.1.6 Moving the game world

To make it look like the character of the game is actually moving it was decided that we would develop an individual class Camera. Camera will handle what part of the game world needs to be rendered depending on the position of the character, it will translate the world positions to the actual positions that needs to be rendered.

2.2 Software decomposition

2.2.1 General

The application is decomposed into the following modules, see Figure 3.
- model, the core object model of the game and the model part of MVC.

- view, the main GUI for the application and the view part of MVC.
- controller, the controllers to the models and vies and the controller part of MVC.
- states, the main controllers of the application, controls the different states of the application.
- map, the classes that handles the game world.
- utils, enumerators and classes like Sound, Camera and Utils, which many of the models and controllers needs access to.

2.2.2 Decomposition into subsystems

The only subsystems are Camera, which handles what should be rendered depending on a position, and Sounds which handles all sounds and music independently.

2.2.3 Layering

The layering is as indicated in Figure 4 below. Higher layers are at the top of the figure.

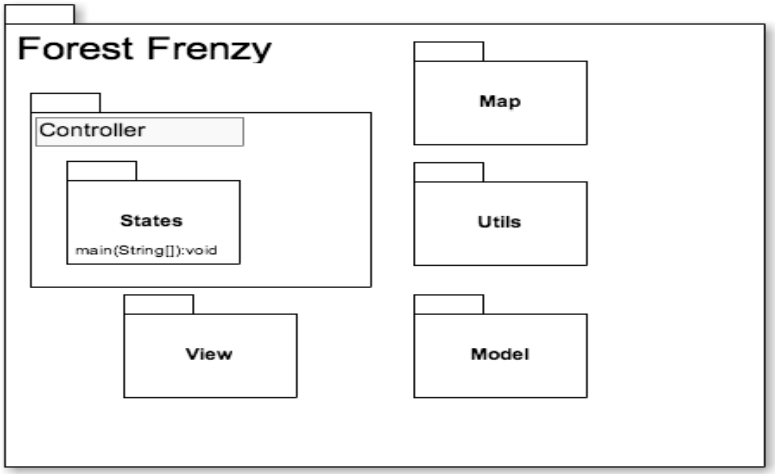


Figure 4: High level design

2.2.4 Dependency analysis

Dependencies re as shown i Figure 5 below. There are no circular dependencies.

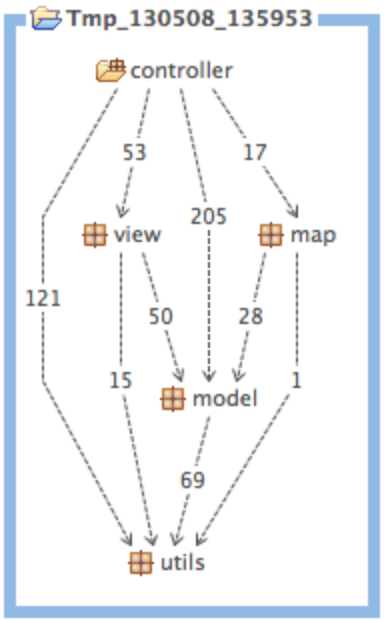


Figure 5: Layering and dependencies analysis.

2.3 Concurrency issues

NA. This is practically a single threaded application. Everything will be handled by Slick2D:s own main thread and apart from that there will only be two timers handling themselves.

2.4 Persistent data management

All persistent data will be stored in flat test files (format, see APPENDIX). The files will be;
- A file for highscores. This is where highscores will be saved and loaded from, this will be reflected in the GUI of the HighscoreState.
- A file for the game world map. This is where the map will be stored and loaded from, it will be directly reflected in the GUI of the PlayState.

(- Several files for images in the GUI, this will be directly reflected by all GUI:s in the application.
- Several files for sounds and music in the application.)

2.5 Access control and security

NA.

2.6 Boundary conditions

NA. Application will be launched and exited as a normal desktop application (scripts).

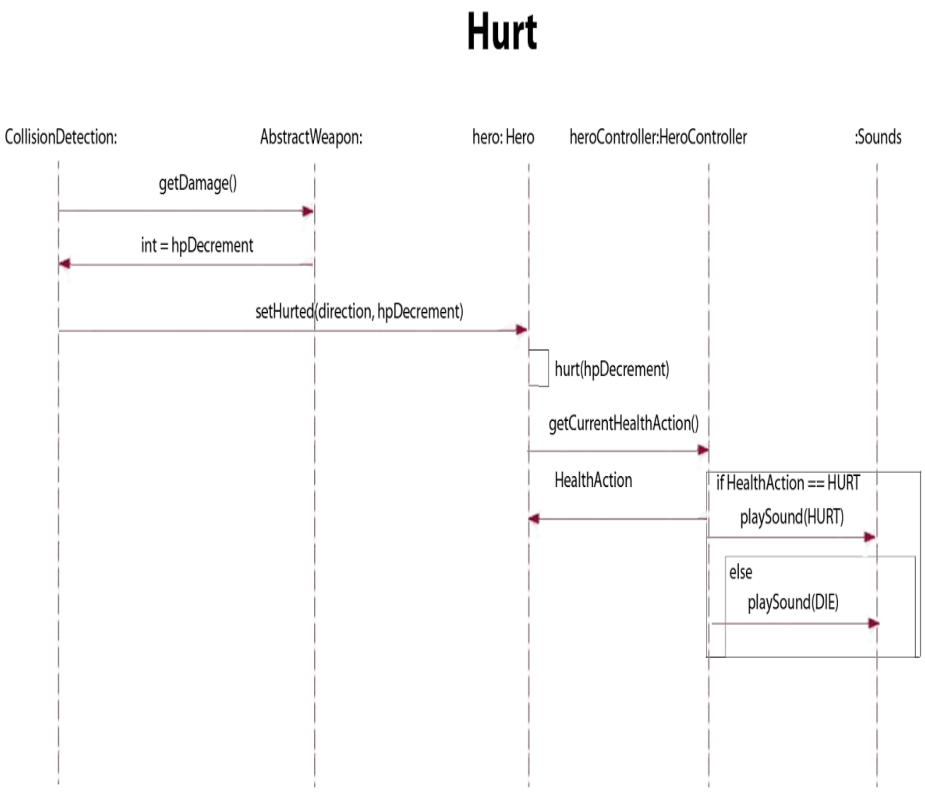
3 References

1. Platform game: http://en.wikipedia.org/wiki/Platform_game
2. MVC: http://sv.wikipedia.org/wiki/Model-View-Controller
3. Slick2D: http://www.slick2d.org/
4. JBox2D: http://www.jbox2d.org/

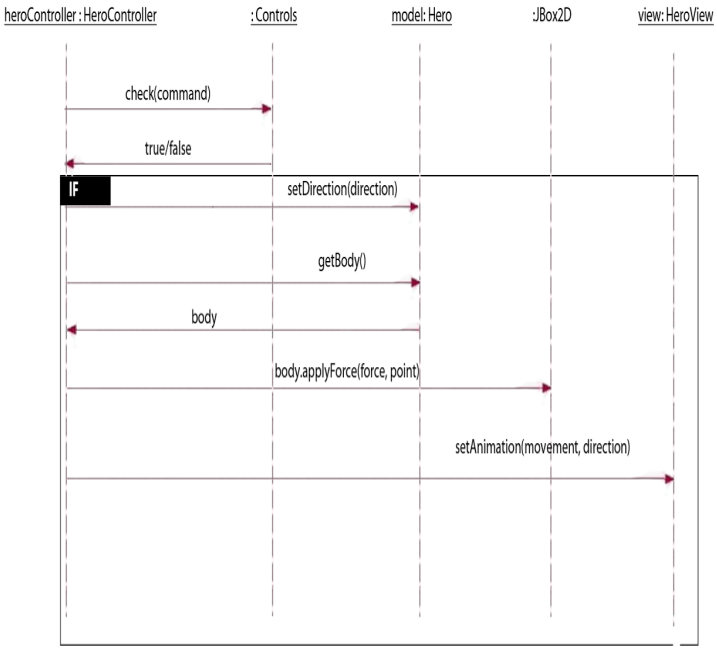
APPENDIX

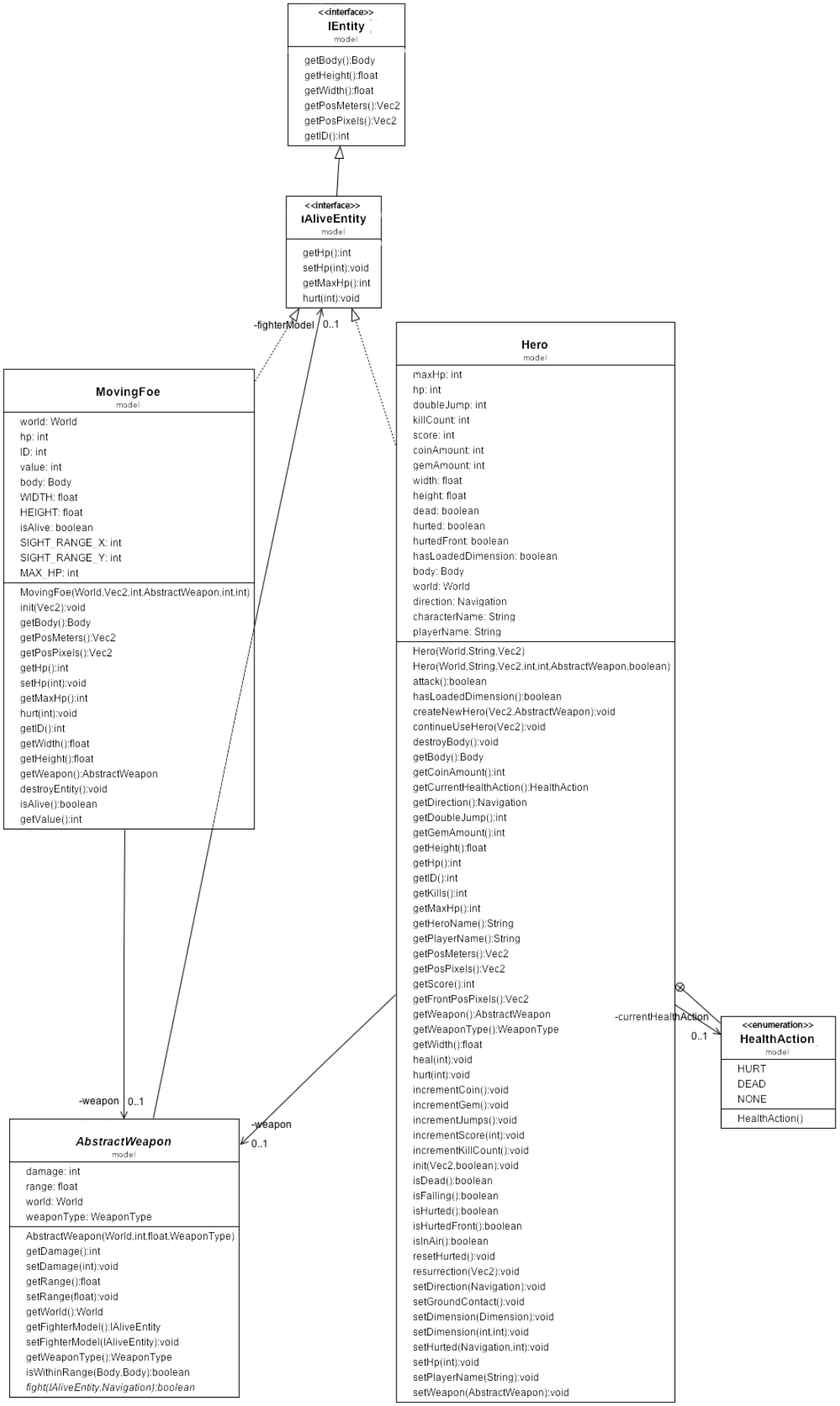
Class diagrams for package
model

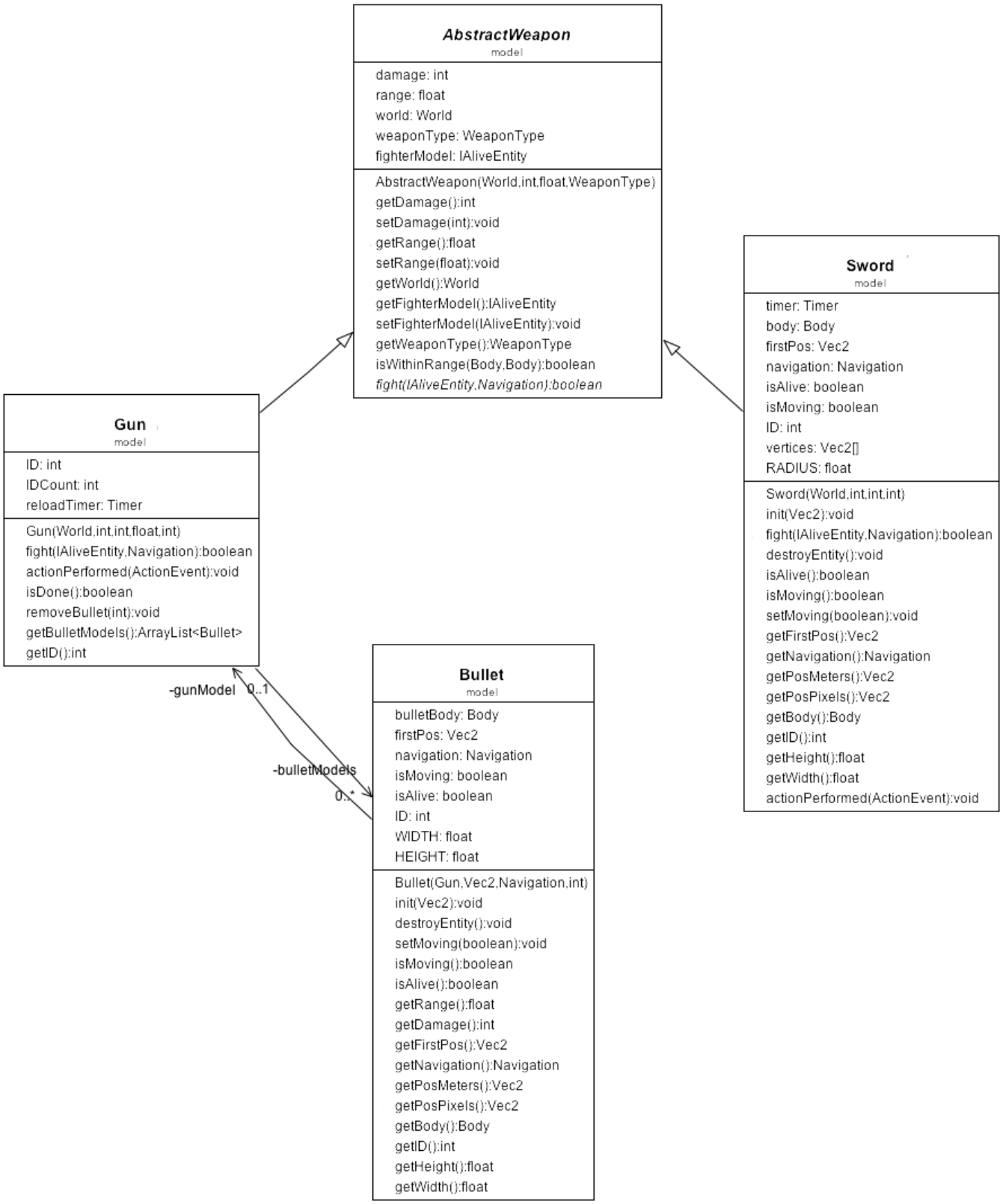
File formats
.dat, a text file format made so that it can't be easily tempered with (highscores shouldn't be modified outside the game)
.tmx, a file format saving a map in a kind of matrix system. Compatible with Slick2D.
.png, an image file format.
.wav, a music/sound file format.

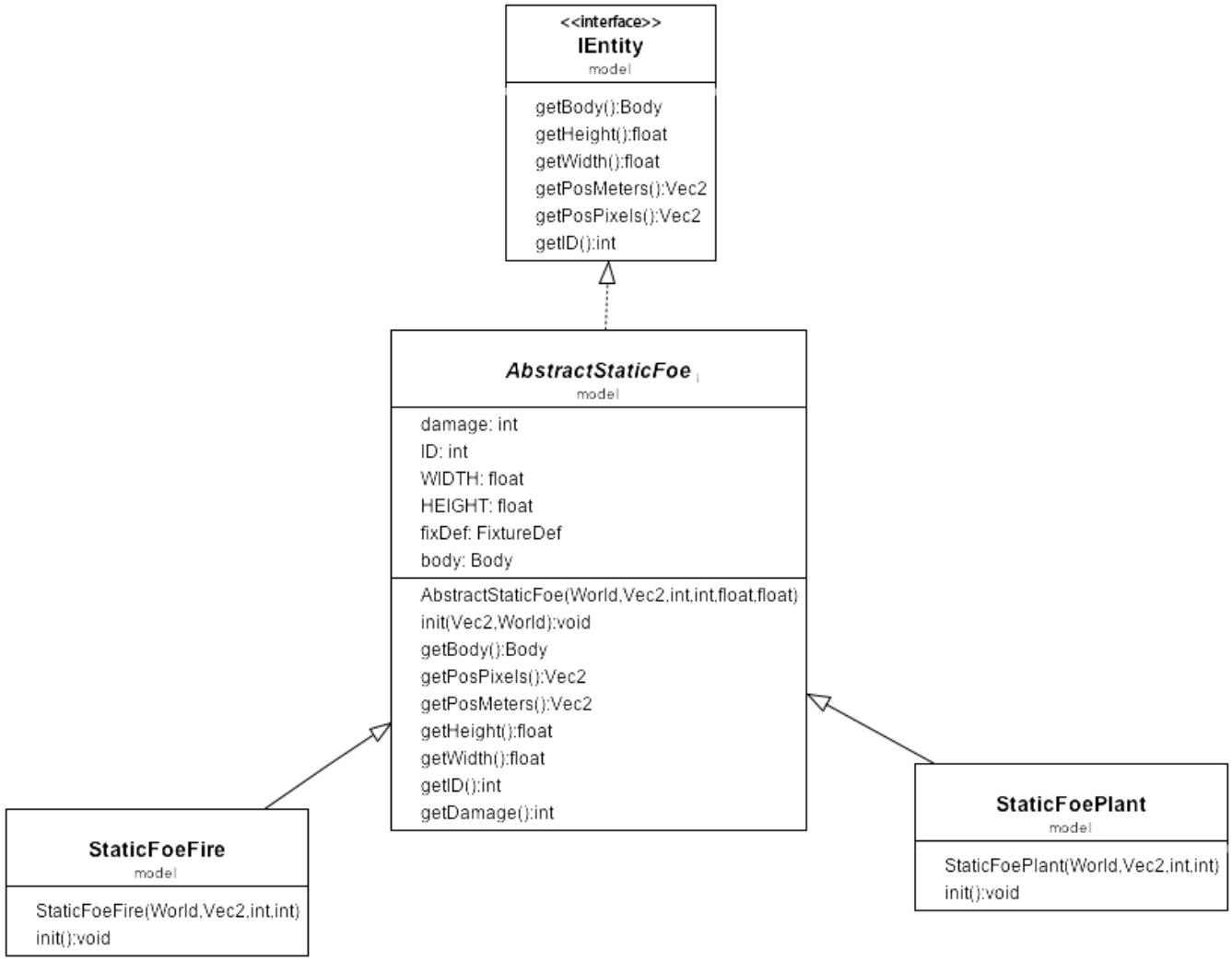


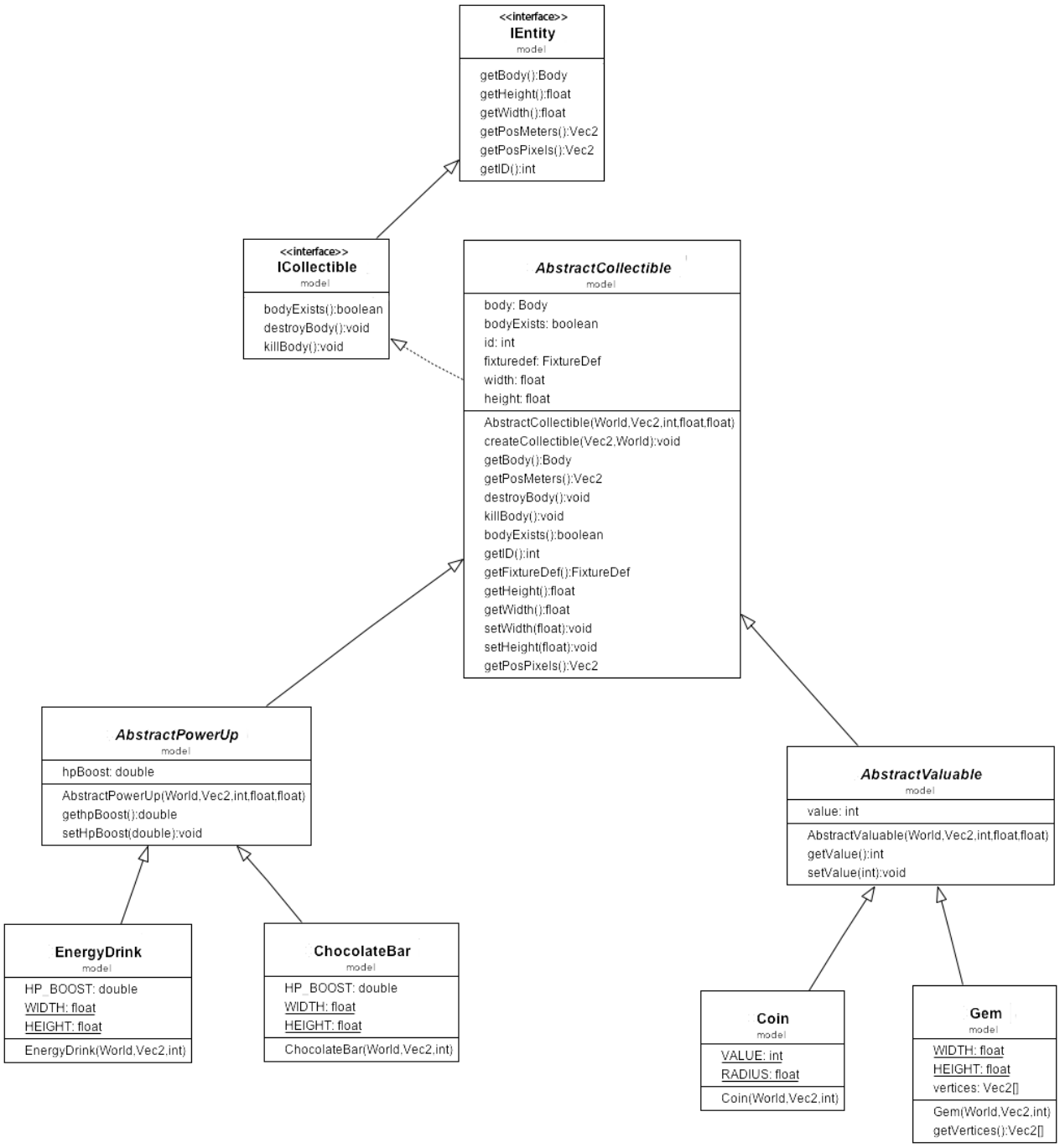
Move

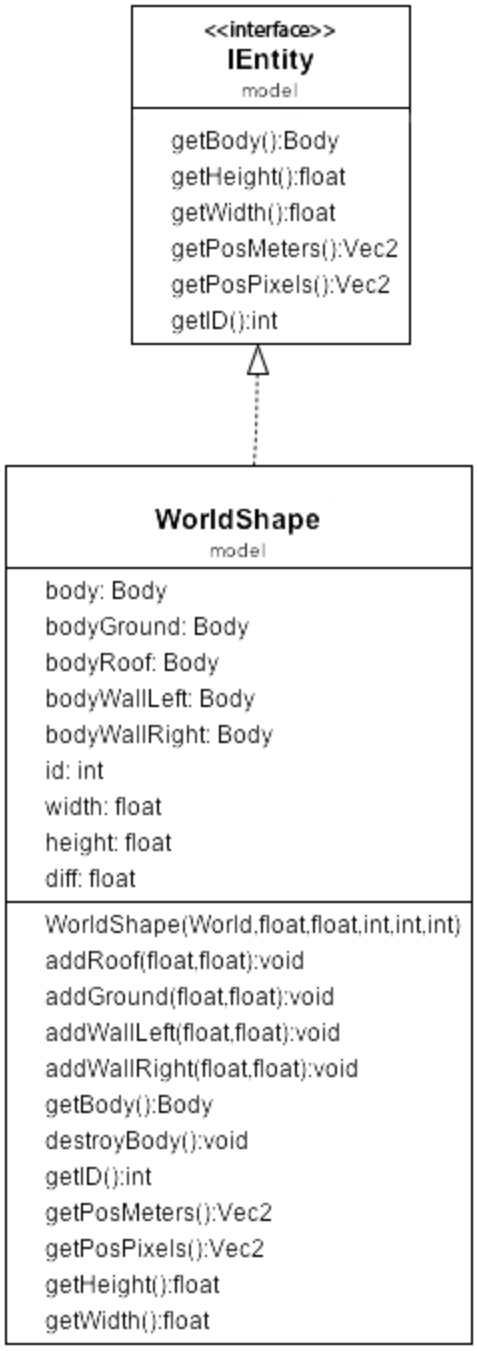

heroController : HeroController
: Controls
model: Hero
:JBox2D
view: HeroView
check(command)
true/false
IF
setDirection(direction)
getBody()
body
body.applyForce(force, point)
setAnimation(movement, direction)


<<interface>>
IEntity
model
getBody():Body
getHeight():float
getWidth():float
getPosMeters():Vec2
getPosPixels():Vec2
getID():int
<<interface>>
IAliveEntity
model
getHp():int
setHp(int):void
getMaxHp():int
hurt(int):void
-fighterModel
0..1
MovingFoe
model
world: World
hp: int
ID: int
value: int
body: Body
WIDTH: float
HEIGHT: float
isAlive: boolean
SIGHT_RANGE_X: int
SIGHT_RANGE_Y: int
MAX_HP: int
MovingFoe(World,Vec2,int,AbstractWeapon,int,int)
init(Vec2):void
getBody():Body
getPosMeters():Vec2
getPosPixels():Vec2
getHp():int
setHp(int):void
getMaxHp():int
hurt(int):void
getID():int
getWidth():float
getHeight():float
getWeapon():AbstractWeapon
destroyEntity():void
isAlive():boolean
getValue():int
Hero
model
maxHp: int
hp: int
doubleJump: int
killCount: int
score: int
coinAmount: int
gemAmount: int
width: float
height: float
dead: boolean
hurted: boolean
hurtedFront: boolean
hasLoadedDimension: boolean
body: Body
world: World
direction: Navigation
characterName: String
playerName: String
Hero(World,String,Vec2)
Hero(World,String,Vec2,int,int,AbstractWeapon,boolean)
attack():boolean
hasLoadedDimension():boolean
createNewHero(Vec2,AbstractWeapon):void
continueUseHero(Vec2):void
destroyBody():void
getBody():Body
getCoinAmount():int
getCurrentHealthAction():HealthAction
getDirection():Navigation
getDoubleJump():int
getGemAmount():int
getHeight():float
getHp():int
getID():int
getKills():int
getMaxHp():int
getHeroName():String
getPlayerName():String
getPosMeters():Vec2
getPosPixels():Vec2
getScore():int
getFrontPosPixels():Vec2
getWeapon():AbstractWeapon
getWeaponType():WeaponType
getWidth():float
heal(int):void
hurt(int):void
incrementCoin():void
incrementGem():void
incrementJumps():void
incrementScore(int):void
incrementKillCount():void
init(Vec2,boolean):void
isDead():boolean
isFalling():boolean
isHurted():boolean
isHurtedFront():boolean
isInAir():boolean
resetHurted():void
resurrection(Vec2):void
setDirection(Navigation):void
setGroundContact():void
setDimension(Dimension):void
setDimension(int,int):void
setHurted(Navigation,int):void
setHp(int):void
setPlayerName(String):void
setWeapon(AbstractWeapon):void
-currentHealthAction
0..1
<<enumeration>>
HealthAction
model
HURT
DEAD
NONE
HealthAction()
-weapon
0..1
AbstractWeapon
model
damage: int
range: float
world: World
weaponType: WeaponType
AbstractWeapon(World,int,float,WeaponType)
getDamage():int
setDamage(int):void
getRange():float
setRange(float):void
getWorld():World
getFighterModel():IAliveEntity
setFighterModel(IAliveEntity):void
getWeaponType():WeaponType
isWithinRange(Body,Body):boolean
fight(IAliveEntity,Navigation):boolean
-weapon
0..1


AbstractWeapon
model
damage: int
range: float
world: World
weaponType: WeaponType
fighterModel: IAliveEntity
AbstractWeapon(World,int,float,WeaponType)
getDamage():int
setDamage(int):void
getRange():float
setRange(float):void
getWorld():World
getFighterModel():IAliveEntity
setFighterModel(IAliveEntity):void
getWeaponType():WeaponType
isWithinRange(Body,Body):boolean
fight(IAliveEntity,Navigation):boolean
Sword
model
timer: Timer
body: Body
firstPos: Vec2
navigation: Navigation
isAlive: boolean
isMoving: boolean
ID: int
vertices: Vec2[]
RADIUS: float
Sword(World,int,int,int)
init(Vec2):void
fight(IAliveEntity,Navigation):boolean
destroyEntity():void
isAlive():boolean
isMoving():boolean
setMoving(boolean):void
getFirstPos():Vec2
getNavigation():Navigation
getPosMeters():Vec2
getPosPixels():Vec2
getBody():Body
getID():int
getHeight():float
getWidth():float
actionPerformed(ActionEvent):void
Gun
model
ID: int
IDCount: int
reloadTimer: Timer
Gun(World,int,int,float,int)
fight(IAliveEntity,Navigation):boolean
actionPerformed(ActionEvent):void
isDone():boolean
removeBullet(int):void
getBulletModels():ArrayList<Bullet>
getID():int
-gunModel
0..1
-bulletModels
0..*
Bullet
model
bulletBody: Body
firstPos: Vec2
navigation: Navigation
isMoving: boolean
isAlive: boolean
ID: int
WIDTH: float
HEIGHT: float
Bullet(Gun,Vec2,Navigation,int)
init(Vec2):void
destroyEntity():void
setMoving(boolean):void
isMoving():boolean
isAlive():boolean
getRange():float
getDamage():int
getFirstPos():Vec2
getNavigation():Navigation
getPosMeters():Vec2
getPosPixels():Vec2
getBody():Body
getID():int
getHeight():float
getWidth():float


<<interface>>
IEntity
model
getBody():Body
getHeight():float
getWidth():float
getPosMeters():Vec2
getPosPixels():Vec2
getID():int
AbstractStaticFoe
model
damage: int
ID: int
WIDTH: float
HEIGHT: float
fixDef: FixtureDef
body: Body
AbstractStaticFoe(World,Vec2,int,int,float,float)
init(Vec2,World):void
getBody():Body
getPosPixels():Vec2
getPosMeters():Vec2
getHeight():float
getWidth():float
getID():int
getDamage():int
StaticFoeFire
model
StaticFoeFire(World,Vec2,int,int)
init():void
StaticFoePlant
model
StaticFoePlant(World,Vec2,int,int)
init():void


<<interface>>
IEntity
model
getBody():Body
getHeight():float
getWidth():float
getPosMeters():Vec2
getPosPixels():Vec2
getID():int
<<interface>>
ICollectible
model
bodyExists():boolean
destroyBody():void
killBody():void
AbstractCollectible
model
body: Body
bodyExists: boolean
id: int
fixturedef: FixtureDef
width: float
height: float
AbstractCollectible(World,Vec2,int,float,float)
createCollectible(Vec2,World):void
getBody():Body
getPosMeters():Vec2
destroyBody():void
killBody():void
bodyExists():boolean
getID():int
getFixtureDef():FixtureDef
getHeight():float
getWidth():float
setWidth(float):void
setHeight(float):void
getPosPixels():Vec2
AbstractPowerUp
model
hpBoost: double
AbstractPowerUp(World,Vec2,int,float,float)
gethpBoost():double
setHpBoost(double):void
AbstractValuable
model
value: int
AbstractValuable(World,Vec2,int,float,float)
getValue():int
setValue(int):void
EnergyDrink
model
HP_BOOST: double
WIDTH: float
HEIGHT: float
EnergyDrink(World,Vec2,int)
ChocolateBar
model
HP_BOOST: double
WIDTH: float
HEIGHT: float
ChocolateBar(World,Vec2,int)
Coin
model
VALUE: int
RADIUS: float
Coin(World,Vec2,int)
Gem
model
WIDTH: float
HEIGHT: float
vertices: Vec2[]
Gem(World,Vec2,int)
getVertices():Vec2[]


<<interface>>
IEntity
model
getBody():Body
getHeight():float
getWidth():float
getPosMeters():Vec2
getPosPixels():Vec2
getID():int
WorldShape
model
body: Body
bodyGround: Body
bodyRoof: Body
bodyWallLeft: Body
bodyWallRight: Body
id: int
width: float
height: float
diff: float
WorldShape(World,float,float,int,int,int)
addRoof(float,float):void
addGround(float,float):void
addWallLeft(float,float):void
addWallRight(float,float):void
getBody():Body
destroyBody():void
getID():int
getPosMeters():Vec2
getPosPixels():Vec2
getHeight():float
getWidth():float