

# ZeroZ documentation

\*Object and Class names/types are in bold.

Overview:

Core classes:

**ZerozGame**: this is the main class to initialize the game, it creates the main menu and holds the value of which level number should be loaded

**MainMenu(ZerozGame gam)**

**MainMenu**: the main menu class creates the main menu, it also creates the game scene class object

**(final ZerozGame gam)**

**GameScene**: this is the main game class, it initializes everything belonging to the level, the level

<b>GameScene</b>	Creates:
	WorldLogic
	LevelManager
	WorldRender
	Camera
	Assets
	World
	EntityManager

**WorldLogic(OrthographicCamera camera, EntityManager entityMan, World world, LevelManager levelMan)**

**WorldLogic**: world logic is the primary thread for calculating logic in the game, it checks for nearby **enemyspawners** and auto-targets the closest enemy the player is facing, it calls update functions within the **EntityManager** including its sub-objects to update projectiles. Much of the entity specific game logic, is passed to the entity manager. **WorldLogic** also includes the **InputHandler**.

**LevelManager(int level, EntityManager actorMan, World world)**

**LevelManager**: Primarily, the **LevelManager** scoures tiledmap ".tmx" tiles for properties, such as "solid" or "item" or "enemyspawn" to create the level. The levels may also contain special tiles which

indicate whether the level is a boss level or a scrolling level (a level where the player is continuously running forward), identifies areas of collision to be passed to box2D. It also contains a small amount of data including the players start location and whether or not the level is complete.

\*

```
WorldRenderer(OrthographicCamera camera, World world, EntityManager actorMan, LevelManager tm, WorldLogic worldLogic)
```

**WorldRenderer:** The world renderer is the primary class for rendering the world, it contains the "sprite batches" an on/off state machine where the program is told to begin/end drawing the games sprites for everything in the game.

```
camera = new OrthographicCamera(viewportWidth, viewportHeight)
```

**Camera:** Camera is a class inherited from LibGDX API, it is necessary for converting screen space co-ordinates into world-coordinates.

**Assets:** This is supposed to be where texture files are kept, so that the game does not reload textures for each instance of an object, however, I have found the LibGDX's **AssetManager** class to have a bug where reloading the game on the phone will cause images not to appear correctly, this is perhaps due to the game not closing properly. Never the less, many assets are kept in the class "**MyAssetManager**" and "**HumanSprite**" which do not use LibGDX's "AssetManager class".

**HumanSprite, MyAssetManager:** Loads textures so they are not reloaded every new object.

*TODO: These 3 Asset Classes should be combined into one class*

**World:** World is Box2D's class for managing the world physics, it includes the function:

```
world.step(timeStep, velocityIterations, positionIterations);  
world.step(1 / 30f, 1, 1);
```

*note: This is currently in WorldRenderer class, although that is probably not entirely appropriate.*

This function maintains the rate and detail of which physics are processed.

```
EntityManager(OrthographicCamera camera, World world,
LevelManager levelMan)
```

**EntityManager:** All the **PawnEntity(s)**, **projectiles**, **destoyables**, enemies, explosions, items and their associated arrays are kept with in entity manager. The **EntityManager** handles a significant ammount of game logic, including taking damage, picking up items and destroying objects.

<b>EntityManager</b>	Creates:
Recieves:	MyAssetManager
camera	DazContactListener
world	ContactHandler
tm (LevelManager)	ProjectileManager (for player)
	ProjectileManager (for AI)

```
public DazContactListener(ContactHandler ch)
```

**DazContactListener** and **Contacthandler:**

The results of collisions are detected through an implementation of box2d's **ContactListener** (**DazContactListener**) is sent to an array handler (**ContactHandler**) which reports on objects that will need to be destroyed or take damage due to collision. I was advised that the **ContactListener** Class should not contain any logic, as it can cause the game to hard crash, being called every collision, and so it quickly sends its data to **contactHandler**, which handles the relationship between collisions and stores them in associated arrays.

```
public ContactHandler(){
    enemiesToDamage = new Array<PawnEntity>();
    projToRemove = new Array<Projectile>();
    destroyablesToDamage = new Array<Destroyable>();
    droneToDamage = new Array<Drone>();
    copterTurretToDamage = new Array<CopterTurret>();
}
```

```
ProjectileManager(int proj_limit, World world, MyAssetManager
assetMan)
```

**ProjectileManager:** the **ProjectileManager** handles **Projectile(s)** and **MuzzleFlash(s)**.

Other Info:

**Drone:** **Drone** Class is currently not working, drone is a different type of flying enemy, I want to find an efficient way to integrate it with the pawn entity class rather than rebuilding everything around it.

**GamePhysics:** Each PawnEntity has its own GamePhysics object calculating physics for it, the collision physics for the characters bodies do not rely on box2d, instead they refer to the Tiled Maps tile properties as given by the **LevelManager**.

**DazDebug:** A class I will put anything I think should help.

**ParraLaxCamera:** A class that helps make the "parralax" scrolling background.

```
public void LoadInputHandler(float viewwidthin, float viewheightin, Camera camerain, PawnEntity zplayerin)
```

**InputHandler:** Handles Input. The Screen width, of any given device is translated into a decimal (between 0.0 and 1.0) and the "buttons" such as jump,move,shoot are placed at intervals such as 0.25,0.5 etc.

```
public void newWeapon(int i)
```

**Weapon:** The weapon class holds the id number of a weapon, there are no plans to have more than 10 weapons in zeroz so an several arrays, 10 datatypes long, is sufficient to hold information for each weapon including rate of fire, lifetime of bullet, damage, shots per shot etc.

```
public Item(float x, float y, int id, String itemType)
```

**Item:** Collision detection for the items is not handled by box2d, it is handled simply by iterating through the item location (when the player is crouching) and seeing if it matches that of the player, I found box2d would often ignore that the player was standing at the location on an item, perhaps because both the player and the item where only sensor bodies, I'm not really sure.

I like the idea of having the player crouch to pickup weapons, when the player picks up a weapon, I like to have the player put their current weapon in the former items places as opposed to loosing the weapon or carrying multiple weapons.

*TODO: Item images are not currently handled by an MyAssetManager, allthough they should be.*

```
public Destroyable(int x, int y, int id, World world)
```

```
public Door(int x,int y,int key, World world)
```

**Destroyable** and **Door**: Every destroyable has a key value, and every door has a key value, when a destroyable is destroyed (bug: currently damaged) the program will iterate through all the doors and open the ones with the corresponding keyvalue to allow the player through.

```
EnemySpawner(float w, float h, String spawnType, int  
spawncountin)
```

EnemySpawner: Simply takes the type of enemy and how many it should spawn, includes only a boolean return function that is called by the EntityManager when the player is near a spawnPoint to decide whether or not it should spawn an enemy.

### Methodology:

Circular arrays: Circular arrays are used for most entities and pretty much everything that comes in bunches, because I don't really like or understand iterators and sets, at least, right now I found them tricky to contend with. So most entities have an ENTITY\_LIMIT, for example:

```
public void createMuzzFlash(PawnEntity entity) {  
    activemuzzflash++;  
    if (activemuzzflash > MUZZLE_FLASH_LIMIT-1){  
        activemuzzflash = 0;  
    }  
    muzzleflash[activemuzzflash] = new MuzzleFlash(entity  
, assetMan);  
}
```

Classes like **PawnEntity** and **Projectile**, include a reUse function:

```
public void reUseEntity(Vector2 actorstart, Weapon weapon){  
    this.weapon = weapon;  
    body.setActive(true);  
    body.setAwake(true);  
    isDisposed = false;  
    isOnLadder = false;  
    isDead = false;
```

```
    isAlive = true;
    worldpos = actorstart;
    health = startinghealth;
}
```

It was suggested to me that I should avoid creating and destroying box2d bodies at run time. This could "in theory" create problems if there were more than the maximum of a certain object appearing at run time, though even a small number like 20 does seem to be sufficient.

**Body:** A body is a box2d collision object, a bounding box, it needs to be assigned to a fixture. Bodies can simply have setActive(false) and setAwake(false) when they are not being used.

**Scene2D:** Scene2D part of libGDX that helps drawing with drawing user interfaces.

#### Performance Tricks:

The batteries on Mobile Devices aren't very good for playing games, it uses a lot of electricity, so I want to implement every trick possible to reduce the energy footprint and of course, help the game play smoothly.

*Polling:* Polling is checking something every so often, as such the worldLogic class includes this function, so, for example it does not update the closest enemy to the player, or updating enemy ai because that's not really necessary.

```
public boolean pollCheck(int v){
    if (loopcount % v == 0){
        return true;
    }
    return false;
}
```

Better yet, I would like to implement a polling Class Object soon, that can take a bunch of operations and give them a tag and execute one of them each frame, an example of this is the way the program checks items. The EntityManager only checks to see if the player is at the location on one item per loop tick:

```

public Item itemAtLoc(Vector2 vec) {
    item_poll++;
    if (item_poll >= item.length)
        item_poll = 0;
    if (item[item_poll] != null) {
        if (Math.abs(item[item_poll].worldpos.x - vec.x)
< 2) {
            if (Math.abs(item[item_poll].worldpos.y -
vec.y) < 2) {
                return item[item_poll];
            }
        }
    }
    return null;
}

```

It might be really cool to keep a time stamp of all the polled items and group them together in equal blocks of time, so each different poll consumes similar processing power to maintain a stable frame rate.

#### Future Implementations:

Something about the game still doesn't feel right and I'm sure that adding things like sound and putting the graphics in the right place will help a little bit, I think we'll need to look into things like moving the camera back out a little bit to make everything smaller and maybe slowing game down as a whole.

Although its fast paced feeling could be the heart and soul of it, there is definatly a fair ammount of playtesting that needs to be done.

#### Melee Combat:

I'd like to have the player kick enemies when they are nearby, its not difficult to impliment I just havent done it yet, sometimes the game seems to have difficulty detecting collision when the player and the enemy are right next to each other, so this will help patch that bug.

#### Secondary Weapon:

The player has one button, quickshoot to automaticly shoot at the closest enemy target, I'd like to test the player with a secondary weapon that activates when the player taps the screen, with something

such as a flame thrower, grenade launcher or a rocket launcher, however we need to make sure this doesn't break the game in times where the player is fighting a specific type of enemy that can't be quick shot, like the copterboss's turrets, or fix it so that the player will autoshoot the turrets when there isn't the danger of an immediate "regular" enemy character.

#### Level Selection:

Currently there is no levelselection, because I was having problems with scene 2d.