GDX Engine

Beginner ’s guide

The document will inform you about Gdx engine and learn how to make games using the engine quickly and stable.

Written by Akemi-san
Gdxengine.uni.me

Introduce GDX Engine Website

About Libgdx framework

When i need a way to Develop games in mobile platforms like Android or iOS, i have seen the libgdx framework and i quickly find acquaintance with how it works. I think it similar to XNA Framework that i have enjoyed for a long time.

Libgdx framework is really useful because it is based on the study of game development cross-platform. That is your game when writing by Libgdx framework, it will be played on Desktop, Web, Android, iOS, and in the future can add other platform! When you code the game use Libgdx framework, you can develop game and debug code on the desktop, after the game completed, you can completely take the game to run on different platforms without or very least the need for modifing source. Libgdx framework optimize your game by design essential elements such as resource management is written in native code. Libgdx framework 's documentation is great and their support forum quickly and enthusiastically. Libgdx framework has many game example, source code, and there are many commercial and non-commercial game has used Libgdx framework.

About Gdx Engine and website

This website contains everything i Developed on libgdx. There is something i call it 'Gdx Engine': D. I have Developed the engine's structure nicely from it could be used in many game projects, varying in Genres and game logics. You can rest assured that the entire Gdx Engine are based on Libgdx framework completely, meaning that every features in Libgdx framework you can apply in Gdx Engine and even more easily.

Gdx Engine was born with the purpose of to help you manage your game source code as you write game use Libgdx framework. Of course you might not need to Gdx Engine but I believe the process of developing your game development process will be a lot easier if you use GDX Engine. Because [The Engine Architecture] of Gdx Engine is very flexible and efficient, and is suitable for both the project scale from small, medium to large commercial projects. Gdx Engine is really just born, so it was pretty much limited in terms of features: as the lack of nice effects, camera smoother or more unique shader. So Gdx Engine still is a long way to grow, so I would need more support from you! Any contribution of you are warmly welcome! Please see the [contribution].

This website will inform you about the GDX Engine, how it works and how to use the engine in your game projects. I tried to do the documentation in detail and easy to understand as possible. Including the architecture of the engine, the articles explains the essential components and step-by-step tutorials, tutorials about the features that Gdx Engine support through the game example which I have prepared. All source code for the GDX Engine, and of the example games are free and you can download easily in the [Download and setup]. You're welcome to mail me on MrThanhVinh168@gmail.com or use the [forum].

If you publish your game using my engine code, all you need to receive the usage's rights is display the logo of engine, It can be laid on your splash screen. Your support will help me by promoting the engine very well.

Contribution

Because GDX Engine is developed in time is not long, so it's still a lot of shortcomings, your support will be an important source of motivation for the development of GDX Engine! I am currently looking for members to continue with me to develop this engine. If you have the following conditions, please do not hesitate to email me on the address MrThanhVinh168@gmail.com I always look forward to the contribution from the you!
+ General Requirements: Passion in the field of game development and open-source game engine
+ Engine Programmer: Improve the coding style of the game engine and create more engine features.
+ Graphic programmer: Fluent OpenGL ES Shading Language to create more effect for engine
+ Document Writer: Create your own games using GDX Engine and write tutorials about your games!
You can also support the for GDX Engine simple by tell your friends about this engine and generate backlinks about GDX homepage Engine. If you can create game use GDX Engine, please let me know via email, I am very happy to hear from you!

About author

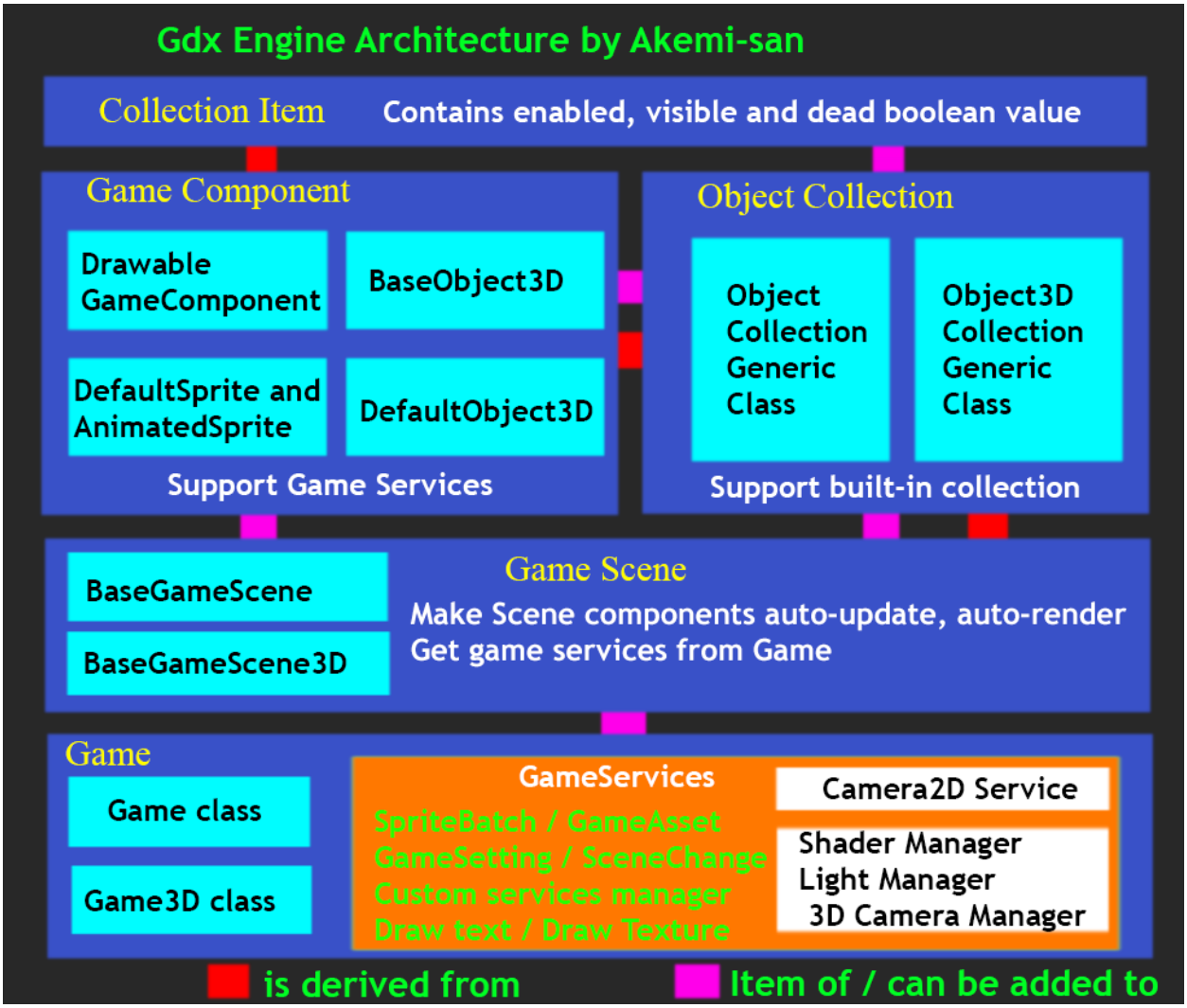
Hi everyone have visited my website. My first name is Vinh and game making is one of my hobbies beside I work in. NET Framework, Java, Wordpress, Web design and I'm working for a out-sourcing software company in my country. Before i start learing Libgdx, I have researched for the XNA Framework and [here] is some my minor project. English is not my native language so my tutorials maybe have some Gramma errors, if you spot anything

wrong, please report me by writing comments. Your comments will be highly appreciated. As you see my nickname is 'Akemi-san' I named that from my favorite anime (cartoon in Japan) If you interest this, you can check the film in [link]. When you need to contact me, you can call me Vinh or 'Akemi-san', both names are fine. You 're welcome to mail me on my email, Maybe i can not reply to all emails you sent to me at once but i will read all of them.

Introduce to GdxEngine

Now i will show you the engine i have refered on [Homepage]. I have separated the engine into many package, and each package will have Their purposes. The full source engine can be downloaded in [here]. The source engine is *brief and simple so you can Develop Gdx Engine by your way Easily.*

The Engine Architecture



Between the rectangle marks connected with red or purple. Order of the collections is from left to right and from top to bottom. Red denotes inheritance. Purple indicates an object may be item / or can be added to another object or not.

Example:
GameComponent => (Red) => ObjectCollection: GameComponent inherited from ObjectCollection!
GameComponent => (purple) => ObjectCollection: GameComponent is item of / can be added to ObjectCollection!
ObjectCollection => (Red) => GameScene: GameScene inherited from ObjectCollection!
GameScene => (purple) => Game: GameScene is item of / can add in Gam eclass!
etc ...

Because of inheritance so **GameComponent** will have boolean variables like *visible* and *dead* from class **CollectionItem**, similar **ObjectCollection** game will also support Game-services like the game component ... etc.

The object is to build specifically for game2D and game3D aims to optimize. Class with string '3D 'in the class name will be used solely for Game3D Development, Game2D Development use the remaining objects of the class. Example: **DrawableGameComponent** for game2D. **AnimatedSprite** for game2D. **DefaultObject3D** for game3D, etc ...

It really very simple. This architecture divide your game into 5 different groups. (Indicates letters of yellow, blue background), there are Collection item, Game component, Object collection, Game scene and Game.

Collection Item

This is the most basic class of engine. No support game services, and most of other game objects in the engine will inherit from this class. It have few boolean variables that indicate:

- **Enabled** (default true): If enable is true, the collection item will be automatically updated as you add it to a object collection, or a GameScene because the object collection will automatically call the update () method of collection items
- **Visible** (default true): If Visible is true, collection item will only **signal** that it should be rendered when it is added to an object collection. Beside, due to the collection item don't have Game-services for rendering, so it can not render, so it does not have the method render () ... The expansion of the collection item class in the game component will also own render function (eg: drawable game component will render for game2D, base object 3D rendering for 3D game)

- **Dead** (default false): A collection of items is dead will automatically set enabled and visible is false, the collection of items that will not be able to auto-update and auto-render again, in addition, if the object has been added to **Game-scene**, Game-scene will automatically remove dead items from its collection. But **Object Collection** will not automatically remove the dead collection item.

Game Component

Object support Game-services, and have all the properties of the collection item. Game-services are a special object is created in the **Game classes.** Game-services contain all services provided by the game, such as taking out the sprite batch, took out the camera, scene management, add an object to do custom services ... Can understand the Game-services is a variable that contains references to the objects will be frequently used during game development. Game-services is shared with all **game components** and extend class from the **game component**. Game-services can be retrieved by method getGameServices (). Unlike Collection item, Game component have own render() method depend on it is 2D Component or 3D Component.

Object Inherited from game object component are four main types:

- **drawable game component**: Has the variable boolean visible: If Visible is true, which collection items will be automatically rendered when you add it into a collection object, or a GameScene because the object collection automatically call render () method of the drawable game component class. The class does not contain available texture or texture region, you have to declare it, then the game can use the services to draw by method drawTexture () or call the service of sprite batch for customization of rendering.
- **Default sprite**: The class extend from the sprite class of Libgdx class framework. Since the sprite contains available constructor permits transmission to a texture or texture region, so you will be very comfortable when using this class. Course because default sprite is among the game component, it completely can be added to any Game-scene or an object collection using the their add... () method. Animated sprite class is extended from the default sprite that supports frame-based animation through an instance of the **Animation** class.
- **Base object 3D**: Object base class for all 3D in your game. Support method renderGL1 () and method renderGL2 (). When you add a base object 3D into a 3D object collection or a 3D Game-scene If android device support OpenGL2.0, method

renderGL2 () will be used to render 3D objects. If android device only supports OpenGL1.x, method renderGL1 () will be used. The use of any method is completely determined by the engine itself, so you do not need to bother.

- **Default 3D objects**: 3D extension of the standard base class object. support Model, Texture of Model and variables store reference to the services of the Game-services for the development Game3D as ShaderManager, LightManager … Use this class is easy, but if you abuse it can lead to wasted RAM. Do the best that you should refrain from creating new objects from this class. For example: class Player can extend from DefaultObject3D. Because the player in the game usually only one instance so the amount of wasting RAM will be negligible.

ObjectCollection

When you use Gamecomponent, engine will turn the call to the update function and render the game component with the each gameloop. If you would like to optimize the program, the object collection is your choice. Imagine you have 100 enemies. Or you create 100 instances of the Enemy class, then add 100 instances in the Game-scene, or you just need to create one instance of EnemyCollection and add 100 enemies to the collection, then add EnemyCollection to Game-scene. Add This process can be done very easily by using the Add method … () of **ObjectCollection class.** EnemyCollection will be responsible for updating and rendering 100 enemies using the method update () and render () method of the object collection inherit from game component. Engine has some object collection as follows:

- **ObjectCollection** class: Using the Collection item as object item. Can customize update and render process of all the items in the built-in collection.
- **SpriteCollection**: Use Default Sprite object item. Default simply calls the update method and render of every Default Sprite object.
- **Object3DCollection**: Using the Collection item as object item. Can customize update and render process of all the items in the built-in collection.
- **DefaultObject3DCollection**: Using to Default 3D object object item. Default simply calls the update method and default rendering of every 3D objects.

Game-scene

When you have created the component and Object games collection, how to use them? Game-scene is the perfect solution for you. Game-scene similar to Object collection because Game-scene inherited from Object collection. Here are a few features for Game-scene:

- Dead items were automatically removed from the collection of Game-scene but not object in the object collection.
- Game-scene represent a scene in the game. For example: Start Scene, Help Scene, gameplay Scene ... while the object collection represents a group of objects in a scene., For example, EnemyCollection only represent all objects "Enemy" in gameplay Scene.
- Game-scene directly receive game-services from Game class and communication the service to other components of the Scene.
- Game-scene is responsible for calling the appropriate method of the Game Object in the Scene, depending on the state of game applications. For example,: when starting activate a Scene, it will call the method initialize () of all components in the scene's collection to initialze all Object item. If the application pause or resume (in android), the current active scene will call the method pause () and resume () to all of the object's item.
- Game-scene has a boolean variable is continueInit. If continueInit is true, at any time you activate a scene, initialize () method of Game-scene will be called. If continueInit is false, the initialize() method is only called the first time the scene is activated.

You can activate a scene (Switch to a different scene) easily using getGameServices().ChangeScene(SceneClass) method.

Example: You have a StartScene, class StartScene extend BaseGameScene class. You are in any scene, you can return StartScene only need to call:
getGameServices().ChangeScene(StartScene.class);

In case you have more than one instance for a class Scene, then you need to use *SceneIndex* to specify the right scene to activate. When you add scenes in the game, the first scene has index is 0, the next scene is similar to 1 and gradually increase.

Example:
/ / Declaration, in your game
StartScene **startScene**;
GamePlayScene **playScene1**;
GamePlayScene **playScene2**;
GamePlayScene **playScene3**;
Scenes to Game, in your game class:
addScene**(startScene);**/ / index of the scene is 0

addScene**(playScene1);**/ / index of the scene is 1
addScene**(playScene2);**/ / index of the scene is 2
addScene (**playScene3);**/ / index of the scene is 3
/ / Change the scene, in any game component!
getGameServices (). changeScene (0) / / Activate startscene
getGameServices (). changeScene (2) / / Activate playScene2

Or can create the final index variable and assign it:

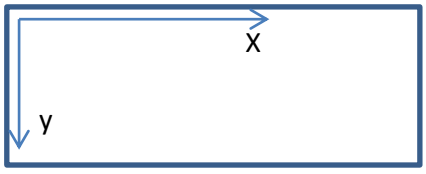
class YourGameClass extend Game {
public static final int **StartSceneIndex** = 0;
/ / some game objects …
}

Then change the scene easily
getGameServices (). changeScene (YourGameClass.**StartSceneIndex**);

Game

When you have more than one scene, you just need to create a new class Extended Game inherited from class Game (for game2D) or class Game3D (for game3D). In the new extended game class, You need to initialize the Game-services and then add your scene to Game and set a scene as active scene. So you have finished games, Engine will automatically update and render the active scene for you! Game-services are quite complex, so I will cover in another article. Game in Gdx Engine should use the y-down coordinate system. The origin (0,0) is located at the top on the left side of the screen. Ox axis runs horizontally, from left to right in the positive. Oy axis running from top to bottom in the positive. Default, Gdx Engine will make game using the y-down coordinate system.

This picture will present the y-down system:



If you want the default coordinate system as of Libgdx framework, simply use method setYdown (false):

```
@ Override
public void create () {setYdown(false); // ...}
```

setYdown () method only makes sense in 2D. In 3D game you don't need to care about this method. setYdown () must be called before calling the initializeGameServices() method of Game

Game-services of GDX Engine

Introduction Game Service

when you develop game with GDX Engine, you will be familiar with a concept very popular in GDX engine, it's the game-services. Game-services are a special object is created in the **Game classes.** The service contain all services provided by Game, as retrieved sprite batch, take out the camera, scene management, add an object to make custom services … You can understand the Game-services is a variable that contains a lot of references to objects will be used frequently when development in game. Game shared services to all **Game Component** and extend the class of **Game Component**. Game-services will be retrieved by method getGameServices() of **Game Component**.

Game-services are separated as **GameService** class for Game 2D (Class Game) and **Game3DService** for Game 3D (Class Game3D). Game3DService inherited from GameService and added some services such as CameraManager, LightManager and ShaderManager that manage all cameras, lights and shaders in any 3D game using GDX Engine. To construct the Game-service, **you must call the initializeGameServices() method** of Game Class.

Game-services allow you to render a texture directly on the screen by using method `drawTexture()`,or render a texture region using the method `drawTextureRegion()` upon the screen. It also allows you to draw any text on the screen using a bitmap font by using the DrawText () method .

Method changeScene () of the Game-services is very convenient when you want to move the screen, refer to the introduction engine architecture, [Game-scene].

You must call **initializeGameServices() method** in your Game class, but it is not required to create every instance for every object services to pass into the method as parameters, you 're allowed to pass 'null' value. If you pass the 'null' value into any parameter in the method, the engine will automatically create an instance instead of null value using proper existing default class.

The service contained two basic types of service, These are **built-in** services, and **custom** services.

Built-in services

This is the services available, including sprite batch, game asset, game setting, scene changing, camera for Game2D and some 3D service managers in game3D. asset and setting are the objects require to create an instance and then use that instance to initialize the Game-services. 3D service managers, of course, is only necessary when you create a 3D game. With 2D game there is no need to.

When you initialize Game-services, you need to pass in an instance of **GameAsset** and **GameSetting**. You can extend from the base class **BaseGameAsset** and **BaseGameSetting** in the engine to generate asset or a special setting for your game, or you can use the default class that engine has built.

To initialize the Game-services, You need to call method initializeGameService ( ) of the Game class, there are two parameters using the default built-in class of Gdx engine.

Example:
//Call in Extended Game class
initializeGameService (new **DefaultGameAsset** (), new **DefaultGameSetting** ());

In game3D, with 3D services manager, you can create an instance of the 3D Managers, and initialize the Game-services. In game3D, there are five parameters

For example:

```
/ / Create managers
CameraManager cameraManager = new CameraManager ();
LightManager lightManager = new LightManager ();
ShaderManager shaderManager = new ShaderManager ();
/ / Initialize settings and assets
BaseGameSetting setting = new DefaultGameSetting ( );
DefaultGameAsset asset = new Asset ();
/ / Initialize Game-service for all scene can use later
initialzeGameService (asset, setting, cameraManager, shaderManager,
lightManager);
```

When you call method initializeGameService, engine automatically call the load…() method of **GameAsset** and **GameSetting** in Game-services to load the asset and setting when the game is starting.

Custom services

is discretionary services that contain references to other objects in the game. For example, you make the asteroid game, in which the player is a ship floating in space. Ship will be destroyed if a collision with Meteor or hit the Enemy's bullet. So you can think the player is a Game-services. the the Meteor and EnemyBullet class can use the player game-service to check for collision with the player. This actually is the ship which gamer controls in the game. For Player can become a services, it needs implement IService interface. Rest assured this is just empty interface to notify Gdx engine that instances of this class can be used as a service in the game.

```
/ / Player class declaration
class Player extend GameComponent implements the IService {/ / your code}
/ / Create a real player that gamer controls:
Player player = new Player ();
// Make player into a service.
getGameServices (). addService (player);
```

You just using method addService () of Game-service so this can turn the player into a services!
To use the Player service , you simply call to getService() method of Game-service object:

```
/ / In the Meteor class, in initialize () method. Get the Player service.
Player player = getGameServices().GetService(Player.class);
/ / Check collision, in the update method of the meteor class:
If (player.collides(this) { / / do some funny stuffs... }
```

getService () method is a generic method, which means you do not need to cast the return object to Player. The method wisely return the Player type, but it will return the data type Player. Similarly if you call getGameServices (). getService**(ABCXYZ.**class); getService ( ) method will return type **ABCXYZ** for you :-D

You need to **get** a service, then **add** service before! If you do not correct the order, the service which will be null and Engine will throw an GdxRuntimeException "Service is not existing...". To be safe, it is best to add services in the constructor of the Scene, and get services in method initialze () of the Game component object.

Default permission for number of using custom services up to ten. If you want to change, before calling method `initialzeGameService()`, you need to specify the number of services that you would like in static variable MAX_SERVICES:

```
Example:
GameService.MAX_SERVICES = 20;
/ / Initialize Game-service for all scene can use later
initializeGameService (asset, baseGameSetting );
```

code above will make your game supports up to 20 Custom services.

GameAsset and GameSetting!

Every game needs two basic features, resource management and storage set and setting.

Game Asset

You already know that game asset is an indispensable component in method initializeGameService () of the Game class, what exactly is the function of the Game Asset?

Asset will manage all the resource in your game. resource is the resource that you use such as texture, texture region to render object2D or covered up a Model3D. Sound and Music to create sound effects, bitmap font used to draw text to the screen, shader to render object3D … **Asset load resource management also dispose resource. Asset provides utility methods to load the resource is quick and convenient. every game asset inherited from class BaseGameAsset, Engine has a class that implements the standard is DefaultGameAsset.**

Every resource should be load when you start the game and should be freed when exit game. Sourcing resource (asset), use resource for game object and dispose resource is done in Game Asset. Years DefaultGameAsset has implement IGameAsset interface and provides methods function to load the texture, load the texture region, load music, load bitmap font resource load … The whole process should be done in the load () method of GameAsset. dispose process should be done in the dispose (). DefaultGameAsset a boolean isLoaded (Default: false) signaling game resource load all or not:

For example:
/ / Declaration

```
	public static Texture BGTexture;
	@ Override
	public void load ()

		{if(isLoaded)
			return;

		BGTexture = loadTexture("spacewar/bg.png",Format.RGBA4444,
true);
		/ / Load for more resource ...
		isLoaded = true;}


	@ Override
	public void dispose ()
		{BGTexture.dispose()
		{// Dispose for more
	resource}
```

Load () method is called when you initilize Game-services and dispose () method is called when exit game. Engine will call this method for you.

GameSetting

This class is responsible for create, load, modify and save your settings in the game, including the game data of the user that you want to save. **Every Game setting legacy inherited from class BaseGameSetting, Engine has a class that implements the standard is DefaultGameSetting.**

Method loadSetting () will be called automatically when you call method initializeGameServices () to load the setting of the game when you start the game. Method saveSetting () will be called when your game is Paused or before exit game. It will save the setting for the next time you start Games.

ModifySetting (int key, Object value) method and getSetting (int key) is an abstract method, no part implements standard so you will have to write them in the extended class if they wish to use this method. You can see an example in [this]

class `DefaultGameSetting` implements the method for the save and load data under as Data Serializabled. You can create a class GameData contains the value to be stored in hardware devices, **GameData** implements theinterface **Serializable** in thepackage. **java.io**then you can use methods loadSetting (Class) and saveSetting (Serializable) of

`DefaultGameSetting` to create new, load or save GameData very easily. Static variable 'savefile' of `DefaultGameSetting` will name specified save file saved on the device's hardware.

Example:

```
DefaultGameSetting.file = ".spacewardata";
BaseGameSetting baseGameSetting = new DefaultGameSetting();
gameData = baseGameSetting.loadSetting(GameData.class);
if(gameData == null)
{
	gameData = new GameData();
	baseGameSetting.saveSetting(gameData);
}
```

the specified code name save file is '. spacewardata' and if load gameData is null (When you first start the game, the game can not save file, loadSetting () method returns null), the game will automatically create save file is '. spacewardata' default and save the data in the GameData class file on hardware device, by use `saveSetting ()` method.

Not only Serializable game data is supported, the object interface **Externalizable** Data will supported similar methods loadSetting (Externalizable) and saveSetting (Externalizable).

Manager Object of GDX Engine!

Overview Manager Object

In Gdx Engine pretty much the Manager, a Manager to manage a group consisting of similar objects but most of the time we will use only one of these subjects. Some typical manager can take an example: **camera, light** or **shader**manager.

Example, your game can have multiple scenes, but in time, you just need a scene. You can not both render "StartScene", rendering "GamePlayScene" same time! Similarly, you can have multiple cameras in a 3D game, FirstPersonShooter Camera, ThirdPersonShooter Camera, FixedCamera ... but in a moment can only use a single camera to observe the 3D object in the game only. You can not both at once both use FPS Camera use TPS Camera. Subjects are being used will be save reference in variables **activeObject,**You view BaseManager.java source for more details. The object Manager birth to provide a solution for managing the camera, the scene, the light, and more.

BaseManager is the superclass of all the manager of engine.Trong then BaseManager has built the basic method as get () ..., setactive ... (), add (), ... BaseManager is generic class so it can be used easily to create higher-level Manager is available or you can create new custom Manager. You only need to specify the class for thevalue **key** andclass type for **value** itswhen creating a new Manager

Example: public class CameraManager extends **BaseManager**<String, **Camera>**{/ / do some stuffs ...} The

above code creates a CameraManager manage all **cameras** based on the key of each camera when be added to CameraManager's collection using the add method ... () of CameraManager

example:

```
//Add Camera Method in CameraManager class
public void addCamera (String key, Camera camera)
        {
                if
               (collection.containsKey ( key))
                        throw new GdxRuntimeException ("Camera key is existing in
CameraManager!");

                        collection.put (key, camera);
        }
```

Manager with **ObjectCollection** engine. **ObjectCollection** for a set of objects interact with each other directly and that is the object can display in GameScene. **Manager** is used for services and often only necessary to set the active object in a certain time while the game is running. **ObjectCollection** use **ArrayList** collection, also **Manager** using **HashMap**. LightManager not need to use to method setactive ... () because in theory all the light in GameScene should be used at the same time rather than using each light individually.

BaseManager implements interface IService, meaning you can add it to the object service game easily.

Step-by-step Tutorial Game2D: Dropcatch

Introduce to Dropcatch game

This article will help you acquainted with GDX Engine. The best way to learn how the game is started to make a real game! Before you start doing game called "Dropcatch". I presume so you have a better understanding of [The Engine Architecture] of GDX Engine, Know about [Introduction Game Service], [Game Asset] and [GameSetting], and [Overview Manager Object].

In Dropcatch two basic types of objects: **drop** are the drops of water fall down from above and **bucker** that catch the drops are falling. If the drop fall on bucker, the drop will disappear, bucker took a drop, and so player (bucker) earn points. bucker can be controlled using the left and right arrow keys, or by touch on the touch screen of the device. After a certain period of time, the drop will be increased in order to increase the difficulty of the game. For example, after 5s game will have 2 drops, 3 drops after 5s… and rise continuously until the player loses (After omitting too many drop)

Documents of Libgdx framework also has a similar game dropcatch this , you can refer to [here]. Tutorials about game Dropcatch will guide you step-by-step. If you are really starting to step into the field of game development, tutorials really useful to you.

Construction Game Asset

GameAsset is required for Gdx Engine and is always the first step when you create a new project game. I recommend create Asset inherit from `DefaultGameAsset.`

```
Public class Asset extends DefaultGameAsset

     {public static TextureRegion dropTexture;
     public static TextureRegion buckerTexture;
     public static BitmapFont bigFont;

     @ Override
     public void load ()

          {if(isLoaded)
               return;
```

```
            dropTexture = loadTextureRegion
(loadTexture("dropcatch/drop.png",Format.RGBA4444, true), new Rectangle
(0, 0, 32, 32));
            buckerTexture = loadTextureRegion
(loadTexture("dropcatch/bucker.png",Format.RGBA4444, true), new Rectangle
(0, 0, 64, 64));
            bigFont = loadBitmapFont("data/font16.fnt", "data/font16.png");

            isLoaded = true;}


      @ Override
      public void dispose ()
            {dropTexture.getTexture (). dispose ();
            buckerTexture.getTexture (). dispose ();
            bigFont.dispose
      ();}}
```

This code creates two static variables to save the texture region for other objects in game is drop and bucker. A bitmap font to draw the score of the player (bucker) on the screen.

Game Asset using the load () method that is called when the game started and dispose() method that is called at the end of the game, dispose() is used to free the memory which contains the resource of the game.

Make Drop object

Drop is a real object in the game, however the number of drop will increase so to completely control the drop, we will use the **Object collection** to store the entire drop of the game in it. You can add objects that inherited from Collection item or Game component into Object collection. But to optimize the game, we will use the drop as a Collection item. Collection item without using the render () method, so we will render () all of the drops in the Object collection of drop class, class named DropCollection. However before create DropCollection class, we need to create the Drop class.

Example: Declare and initialize the position, and define the boundaries for the drop:

```
public class Drop extends CollectionItem

      {private Rectangle bound;
```

```
final Vector2 position = new Vector2 ();

public Drop ()
      {super();
      resetPosition ();
      bound = new Rectangle(position.x, position.Medicine,32,
32);}
void resetPosition ()
      {position.y = 0;
      position.x = MathUtils.random(Gdx.graphics.getWidth
());}
public Rectangle getBound ()
{bound.
      x = position.x;
      bound.y = position.y;
      return bound;}
```

You can see the drop variable named **bound** of type **Rectangle.** bound is a rectangle surrounding the object drop, so the position of the bound will coincide with the position of the drop. On the other hand the length and width of a drop texture is 32 pixels (file drop.png), so the bound will be the size of the other two sides is 32. So method `getBound()` always returns exactly a rectangle surrounding the object drop in game scene.

Method resetPosition () will reset the coordinates of the drop, the coordinates y = 0 shows that the drop will appear on the top screen and have a random x coordinate between zero and the length of the screen (random 0 => The ScreenWidth)

Example: Updating drop

```
@ Override
public void update(float gameTime) {
      position.y + = 5;

      if(position.y > GDX.graphics.getHeight ())
      {
          resetPosition();
      }
}
```

Drop will drop from top to bottom, that is position-y drop will vary with each game loop. To make this change, you need to Override the method update () method of the Collection

item. As the above example, position-y will increase by 5 pixels per GameLoop, this makes water move down. Tha method have code to test the position of drop, if the drop across over the screen, we will reset its position.

Construct DropCollection objects

AfterDrop class, obviously you will need a class DropCollection to manage all drops in the game.

Example: the DropCollection class

```
public class DropCollection extends ObjectCollection <Drop> implements
IService{

	private float timer;

	public DropCollection (IGameService services)
	{
		super(services);
	}
	@ Override
	public void initialize (){
		timer =5f;
	}
	/ / ...
```

As you can see, DropCollection inherited from Object collection and implements interface IService. This Means DropCollection can been added to GameService which all objects are supported games services will accessible to the DropCollection from anywhere.

Timer is a float variable, we assume the following 5s will be adding one new drop. if you initialize the timer = 0f, you will have to wait for 5 seconds when the beginning of the game to get the first drop. Set timer = 5f help you not have to wait 5 seconds before seeing the first drops of water: D

Method initialize () is always called when the gameplay Scene is activated. Please rest Assured that this method will Surely be called to at least once when you activate the scene. The code to create the Scene and activate Scene I will mention later.

Example: update and render DropCollection

```
	@ Override
```

```
	public void update(float gameTime)
		{/ / call the update () method of all drops
		super.update (gameTime);
		/ / Increase timer
		timer + = gameTime;
		/ / create a new drop each after 5 seconds for make the game
more difficult
		if(timer> 5f)
		{/
			/ reset the timer
			timer = 0f;
			/ / add new drop into collection
			addItem(new Drop());
		}
	}

	@ Override
	public void render(float gameTime) {
		for(Drop b: objectCollection) {
		if(b.isVisible ()){
			getGameService ().drawTextureRegion
			(Asset.dropTexture,b.position.x,b.position.y);
		}
	}
```

Section code `super.update(gameTime);` in update() method automatically call all update method () of all enabled items in the collection.render part is very simple, loop all the drops and render the textureRegion in the game asset

Construct object Bucket

Bascause the player directly controls bucket using the keyboard or touchscreen so we can call the bucket is 'Player'.

Bucket need to be rendered to the screen through texture or texture region. Because DefaultSprite class has supported this feature, so to save development time, we will create a Bucket new class Inherits from DefaultSprite

```
public class Bucket extends DefaultSprite {

	public Bucket (IGameService services, TextureRegion region)
	{
	super(services, region);
```

```
	}
}
```

Bucket still need to make up the necessary value, it is very simple:

```
	/ / attribute for player game state
	public int score = 0;
```

score is present the player's score received. Once Gamer control bucket catch a drop, the score will increase by one.

Bucket needs to be initialized, you see:

```
	/ / store reference to DropCollection service.
	DropCollection drops;
	@ Override
	public void initialize ()
		{
			score = 0;

			drops = getGameService ().
getService(DropCollection.class);
		}
```

In the code above, bucket is set in the middle and bottom of the screen length. drops variables stored reference to variables dropCollection in the the the gameplay Scene. Drops were retrieved using the method getService () method of the object Game Service.

Example: render and update Bucket

```
	@Override
	public void render(float gameTime) {
		super.render(gameTime);

		getGameServices().drawText(Asset.bigFont, "Score: " + score,
300, getGameServices().getWindowSize().y - 60);
	}
	@Override
	public void update(float gameTime) {

		// store position-x from last frame
		float oldXPosition = getX();
		// control bucker using keyboard
		if(Utils.isKeyDown(Keys.RIGHT))
		{
			setX(getX() + 10) ; //move bucker to right
```

```
            }

            if(Utils.isKeyDown(Keys.LEFT))
            {
                  setX(getX() - 10); //move bucker to left
            }
            // control bucker using touchscreen
            final Vector3 touchPos = new Vector3();
            touchPos.set(Gdx.input.getX(), Gdx.input.getY(), 0);
            getGameServices().getCamera().unproject(touchPos);
            setX(touchPos.x - 64 / 2);
            //if bucker out of screen bound, restore its old position
            if(getX() < 0 || getX() > Gdx.graphics.getWidth()-
getRegionWidth() )
            {
                  setX(oldXPosition);
            }
            //Check collides with all drops
            for(Drop drop : drops)
            if(getBoundingRectangle().overlaps(drop.getBound()))
            {
                  //increase score if bucker catch a drop, reset drop's
position
                  score++;
                  drop.resetPosition();
            }
      }
```

The render () method is very simple, because the class DefaultSprite itself can render the texture or texture region. We add code to render a text containing information about the score of the player the screen.

Method update () contains code to check bucket collision with the bound of the screen and the drops in PlayGameScene.

Overlaps() method of the class Rectangle checking the collision. Collision Between two objects will take place when two their Rectangle overlaps each object, and the the overlaps method () will return true.

Make class GamePlayScene

After create the drop and bucket, you should take it into a GameScene. Above I mentioned the concept "GamePlayScene" Really this is the Scene of the Game and obvious it must

Inherit from BaseGameScene class of GDX Engine. If your game is more professional, it should be many scene as "StartScene", "HighScoreScene", "HelpScene" … But to make this tutorial simple, we will only create "GamePlayScene" to make the game can play at once.

```
Public class GamePlayScene extends BaseGameScene {

      Bucket bucket;
      DropCollection drops;

      float timer = 10;

      public GamePlayScene (IGameService gameService) {
            super(gameService,true);
            / / initialize for player
            bucket = new Bucket (getGameServices () and
Asset.bucketTexture);
            addDrawbleObject(bucket);
            / / initialize for more objects...
```

you see, very simple GamePlayScene. You declare the object (bucket and drops), construct them and add them to Scene by method `addDrawbleObject()` So bucket and drops will be auto-initialize, auto-update and auto-rendering therefore you do not need to call explicitly these methods in class GamePlayScene! You rest Assured because this time GDX Engine called the method initialize (), update () and render () of the items in GameScene rather for you.

In Bucket class, we used the "DropCollection" service. This service is not available in first the place so you must do two steps to make DropCollection service available.

1. Ensure DropCollection execute interface IService
2. uses method `addService ()` of Game object services to make the drops into a game object services and all support services that will use the game DropCollection this service.

The code below will initialize for bullets and make bullets using as game service.

```
            //initialize for bullets
            drops = new DropCollection(getGameServices());
            getGameServices().addService(drops);
            addDrawbleObject(drops);
```

Complete the game with class CatchDropGame

If you really have reached this step, congratulations! you almost had your first complete game using GDX Engine! Here the code of CatchDropGame class

```
public class CatchDropGame extends Game{

	@Override
	public void create ()
	{
		/ / initialize settings and assets
		BaseGameSetting baseGameSetting = new DefaultGameSetting ();
		IGameAsset asset = new Asset ();

		/ / initialize game service for all scene can use later
		initializeGameService (asset, baseGameSetting);

		/ / initialize the scene (s) and add them to scenes collection
		IScene scene = new GamePlayScene(this.getGameService ());
		addScene (scene);
		/ / Active the gameplay scene , initialize the active scene
		getGameService (). changeScene(classGamePlayScene.);
```

Create () method of the Game class is Called as soon as the game is started. You create the Game Asset and game setting to initialize the game-service, you create the scene, add scene in the game, and activate the scene you wish by calling the method changeScene () from the game services. Since you have initialized Games service, The game services is not null and you will not have common exception "Null Object Reference ..."

Advanced 2D Game Tutorials: SpaceWar

Introduce to game SpaceWar

When you read this article, I presume it's you complete the tutorials about basic game: catchdrop in a [Introduce to Dropcatch game]. This tutorial Refers to the feature, not a step-by-step as tutorials on game DropCatch.

During this tutorial, I will cover more advanced Techniques when you program a 2D game that GDX Engine support . It Include about the particle effects, load and save the new setting from the serializable data, scrolled background, to create and display a optional menu, limit using the keyword **new,** create animation ...

This is a real game and there is a number of factors to attract players. In the game you control a ship traveling in the universe. You sail the ship, Avoiding the Meteorites in space to not have them destroyed. You are equipped to actively missile can destroy the meteors that is flying too close. When the asteroid is destroyed, a nice explosion effects will occur to illustrate the clash the between the missile and Asteroids.

Creating particle effects. (File gamePlayScene.java)

GDX engine has class ParticleEffectManager help you Easily manage your particle effects in the game. You can refer to the article [this] to understand the particle effect and particle editor is.

To use particle effects in the game, you need to create an instance of ParticleEffectManager class in a Scene that used as Playing Scene, Add the ParticleEffectManager to Scene's collection and to Game Services.

```
// initialize for effects
effects = new ParticleEffectManager(getGameServices());
addDrawbleObject(effects);// Add effects to Scene's collection
getGameServices().addService(effects);  // Add effects to the game service.
```

ParticleEffectManager class is Manager of all particle effects, you need to add new particle effect to the manager , for example:

```
effects.addParticleEffect("explosion", "Spacewar /explosion.pp",
"spacewar");
```

parameters above: string key of particle effects, particle effect files (Created by Particle Editor tool of Libgdx), and the name of the folder containing the image that is used by particle effect.

To use particle effect in a given game object component, you need to take the corresponding service of ParticleEffectManager to variable reference *effects*:

```
effects = getGameServices (). getService(ParticleEffectManager.class);
```

Then you call the method invoke () to generate effect

```
effects.invokeEffect("explosion",x, y); / / In Player.java file
```

Particle Effects that have key "explosion" will soon be generated in the x and y coordinates passed.

load and save settings from serializable data (File GameData.java)

Now I will show you how to save and load settings from a any serializable object.

First you need To have a class that contains the data you need to save , of course it must have serializable interface.

```
public class GameData implements Serializable {
	public int[] highscores = {50, 40, 30, 20, 10};
	public boolean PlaySound = false;
```

The class above will have two setting is **Highscores** and **PlaySound**. You absolutely add value as you wish such as playerLevel … playerHealthy … CurrentPosition …

You can load gameData from the saved file or create new save file using the default values in the GameData class if the save file is not created. If the file is not existing, loadSetting () method will return null. *GameData* variable is a static variable in your Game class:

```
DefaultGameSetting.savefile = ".spacewardata";
BaseGameSetting baseGameSetting = new DefaultGameSetting();
gameData = baseGameSetting.loadSetting(GameData.class);
if(gameData == null)
{
	gameData = new GameData();
	baseGameSetting.saveSetting(gameData);
}
```

when you need to save the setting, simply call the game services, game settings and save:

```
getGameServices (). getGameSetting (). saveSetting (SpaceWar.gameData);
```

In above example code, saveSetting () method Saves the information from static turn gameData to the saved file named "**spacewardata**", or create a new file if it does not already exist. gameData is required to make sure that not null.

Creating scrolled background (file Background.java)

technique of creating scrolled background we'll render 2 texture background, this two background will move at the same speed in the same direction, if there is a background

beyond the border of the screen, it will be reset to the position so it always would be located in front of the other background and so on. Two background's position will be exchanged each other.

You need a variable to store the value is calculated position for the background as well as defined time to reset position of background:

```
        screenheight = GDX.graphics.getHeight ();
        screenwidth = GDX.graphics.getWidth ();
        / / Set the screen position to the center of the screen.
        screenpos = new Vector2 (screenwidth / 2, screenheight / 2);
        / / Offset to draw the second texture, when necessary.
        texturesize = new Vector2 (getRegionWidth (), getRegionHeight
    ());
        setX(screenpos.x-screenwidth/ 2);
```

update the location and render the background:

```
    @ Override
    public void update(float gameTime) {
        screenpos.y + = gameTime * 100;
        screenpos.y = screenpos.y % getRegionHeight();
    }
    @ Override
    public void render(float gameTime)
    {   // Draw the texture, if it is still onscreen.
        if (screenpos.y < screenheight)
        {
          setY(screenpos.y);
          super.render(gameTime);
        }
        // Draw the texture a second time, behind the first,
        // To create the scrolling illusion.
        SetY(screenpos.Y - texturesize.Y);
        super.render(gameTime);
    }
```

Because the background is a drawable game component, than you absolutely add it to the collection of the game scene. However the order when you add objects in the scene will decide the order in which the object is to update and render. Their Principle is "First-in first-out" Means that the first object to be added will be the first object to be updated and render. Because this is a background, which means than you need to render the background

first then render to the other object. If you do the opposite, the background will render in fornt of other objects and background will completely cover up the other objects that have been rendered before.

```
// Background is the first the object added into scene
addDrawbleObject(new Background (getGameServices ()));
// add other stuffs
addDrawbleObject(player);
addDrawbleObject(bullets);
```

Create a optional Menu (File TextMenu.java, TextMenuManager.java)

To create the menu, you need to create two your own classes for MenuItem and MenuManager.

```
public class TextMenu extends CollectionItem implements IMenuItem {

	public String text;
	public Vector2 position;
	public boolean selected = false;

	public TextMenu(String text,boolean select, Vector2 position) {

		this.selected = select;
		this.position = position;
		this.text = text;
	}
```

MenuItem simply extends Collection items and have interface IMenuItem as the above code is . Since this MenuItem using bitmap fonts to render text content for the MenuItem, it needs the text String property. If you want to use a **texture** to render the MenuItem, you simply replace the **String text;** with a variable like **Texture menuTexture;** and change the method **getContent()** return **texture** value**.** *selected* variable Signal the user had selected this item or not.

In TextMenuManager class, you only need to override the render () method. Since this is the text menu, so using the render () method is as follows:

```
	@Override
	public void render(float gameTime) {
		for(TextMenu item : objectCollection)
```

```
                if(item.isSelect())
                {
                     getGameServices().drawText(Asset.bigFont,
item.getContent(), item.getX(), item.getY());
                } else
                {
                     getGameServices().drawText(Asset.smallFont,
item.getContent(), item.getX(), item.getY());
                }
     }
```

If you are doing a texture menu, simply replace the DrawText method () by method drawTexture () method or drawTextureRegion () from Object Game Services and render your menu item.

limit usage of the new keyword

you would normally use **new** keyword when creating a new object, you call the **new** keyword before a constructor of the object you want to "new" and then you can use the object it comfortably without having a exception like "Null Object Reference ..." But what is behind the **new** keyword?

You knew Java is managed code, which means instead of having to hand the memory for the object you want to create as in native code, Java Virtual Machine (JVM) will automatically allocate memory and recall memory for you, you just need to code code code class, create new object, and then use that an object, and ... not anymore! JVM will automatically freeing memory when it is necessary (for example: lack of device memory). But it would be wiser if you have Capable of Reducing Wasted memory possibly arise in the game, and the easiest way is not to abuse the variable declaration as well as the new keyword.

When you **new** a object, that is, the JVM will take RAM to store the new object, and you do not have any way to dispose of objects, you just can only wait for the JVM automatically dispose the object. To limit the negative side, objects containing resource as texture, the texture region, Music... in Libgdx was built by native code to optimize program, and can easily and dispose object using the method dispose () of the object to free memory. In GDX Engine, Because it is completely based on Libgdx framework so you can totally do the same.

In spacewar game I have come up with a solution to refrain "new object". You see in class BullectCollection:

```
public void addOrRecycleBullet(Vector2 position)
	{
		//recycle dead bullet if the dead bullet is available
		for(Bullet b : objectCollection)
		{
			if(b.isDead())
			{
				b.setDead(false);
				b.position.set(position);
				return;
			}
		}
		//add new bullet to collection
		Bullet b = new Bullet(position, -5);
		b.initialize();
		addItem(b);
	}
```

When the player presses the space key, a bullet will be emitted from the player and only move Vertically upwards. This bullet can created by keyword **new** or recycled from a dead bullet (bullet can turn dead = true). When bullet recycled, we no need to use the new keyword and therefore not have any more space on the RAM to be assigned for store the bullet. With a the dead bullet, we moved the dead bullet out of the screen, so the dead bullet will not cause any effect on gameplay. When you setDead (true) for a collection item, meaning that enable and visible object of which would be false. Object will not be automatically updated and rendered by Game scene, so it also reduces the CPU load.

For example: method setDead () of the Bullet class:

```
public void setDead(boolean value) {
	super.setDead(value);

	position.x = -100;
	position.y = -100;
}
```

Creating Animation

Here is rotated croped animated textures of Meteor, illustrates the animation of meteor in the file asset/ spacewar / sprite.png. You can see the animation of 8 frames, carry the rotate of the meteor, of course you can render this animation by directly rotate the sprite batch,

no need to frame-based animation. However, to illustrate how to create animation, we will not rotate with the sprite batch but we will use the texture animation below.

Easiest way to create your animation on a object is make the object Inherits from class animated sprite, You need an instance of Animation class passed into the constructor of Meteor:

For example**:** public class **Meteor** extends **AnimatedSprite** {

```
public Meteor(IGameService services, TextureRegion region, Animation
animation) {

            super(services, region, animation);
            resetPosition();
      }
```

The Animation class is actually very simple, an animation consists of three variables:

```
public Animation(boolean isLooping, float frameTime,int numFrame) {
            super();
            this.numFrames = numFrame;
            this.isLooping = isLooping;
            this.frameTime = frameTime;
      }
```

isLooping Controls the ability to loop the animation, *time* indicate the during time for moving on to the next frame and the *numFrames* indicate the total number of frames in the animation.

To control the animation mobe in sole discretion, you just use method playAnimation () of the animated sprite class and pass in the animation object desired. The constructor itself play the animation from animation parameter of AnimatedSprite 's constructor that you pass in from the outside. Method playAnimation () will directly change the animation of the object. You can specify the new region on textures in method playAnimation () to correspond the animation that you want to play.

Animated sprite has two events to control your animation easy, that is onFrameChanged () Occurs when the frame of the animation changed and is onAnimationChanged () Occurs when the animation of the animated sprite changed.

For example: Meteor changing region on the texture if the frame changed:

```
@Override
public void onFrameChanged() {
  setRegion(getRegionX(), 50 * this.frameIndex, getRegionWidth(),
getRegionHeight());
}
```

If you notice in sprite.png files, you will see that the new frame in the animation of the meteor is always below of the current frame, but our position-y 50 pixels apart. The first frame position-y is 0. So the code above will be controlled to animation takes place in accordance with our intentions.

When you write the update() method for Meteor, you must keep the update() method from the Animated Sprite class by remaining the code "**super.update (gameTime);"** to make object update animation.

Basic Game3d Tutorials: SpaceWar 3D

Introduce to game SpaceWar3D

In the the the tutorials, I will show you how to make the 3D version of game spacewar. I presume you've done [Introduce to game SpaceWar]. In the 3D version of spacewar, we will replace the **meteor** (capital only move randomly) into the **enemy** that is the UFO aliens, They can move, change direction and firing toward the player. Most of the resources in the game we will by this time taken from the game Droid Invader 3D of libgdx, you see [here]. A spectacular explosion will take effect to describes the collisions of objects in the game.

Principles of Game scene, Collection item and object collection, asset and the Background in similar spacewar3D in the the The coal spacewar so we will not repeat more.

Game Optimization

With 3D Game, the optimization is extremely IMPORTANT because so much of the equipment, especially portable devices is limited the 3D graphics processing Capabilities. So if you want your game to be played on as many devices as possible, then you must optimize for the game.

If you use game engine to make the game, the positive is really you develop game very fast but together with that, if you have not used properly engine will result in a waste of memory and CPU load increases. Of course GDX Engine is no exception.

Class **DefaultObject3D** of GDX Engine was built very Carefully. It allows you to render Model, support translate, rotate or scale the object in the three axis x, y, z and has full support for programmable shaders simulate light shining on the object. But Because it contains many features so i can not call this class is optimized for gaming. Therefore you should refrain from creating new objects from this class. Instead, with the large number of objects, such as bullets and enemies, I recommend using **CollectionItem** and **ObjectCollection** to optimize the game instead of just using **DefaultObject3D** and add the **object** to the **scene** directly. , However, DefaultObject3D could be used easily to create the model of the player, because the player needs more features that really is supported in DefaultObject3D. On the other hand you usually only need a single player in the game. Thus wasting memory and CPU load is negligible.

Make Bullet and BulletCollection

purposes [Game Optimization], we're going to make bullet inherited from collection items and make BulletCollection inherit from the the object collection.

In Bullet, only Those attributes needed for each new separate bullet is declared. Resource for the bullet as Model or Shader'll Just be declared or called in BulletCollection to limit the waste of RAM.

```
Public class Bullet extends CollectionItem {

	public Sphere sphere;
	public final Vector3 position = new Vector3 ();

	public Bullet (Vector3 position, boolean isPlayerBullet) {
		this.position.set (position);
		this.isBulletOfPlayer = isPlayerBullet;
		sphere = new Sphere (position, 0.2f);
	}
```

sphere variable used to Simulate a sphere that cover globally the bullet in the 3D space. The sphere objects will help us check collisions in the 3D space by using the overlaps () method of the class. The technique here is the same as in the Rectangle class.

Next, declare class BulletCollection too simple:

```
public class BulletCollection extends Object3DCollection <Bullet>
implements IService {
	public BulletCollection (IGame3DService services) {
	super.(services);
```

You see BulletCollection has overridden methods to render as renderGL1 () and renderGL2 (). In these methods, the entire bullets will be rendered on the screen:

```
	@Override
	public void renderGL2(float gameTime)
	{
		//set color shader is active to render bullets
		shader.setActiveShader("color");
		//call begin() method here to reduce number of the method
calling that lead to better performance
		shader.begin();
		//render all bullets if the bullet is not dead
		for(Bullet obj : getObjectCollection())
			if(obj.isVisible())
			{
				if(obj.isBulletOfPlayer)
					shader.setUniformf("u_color", 1f, 1f, 0.5f,
1f);
				else
					shader.setUniformf("u_color", 0.5f, 1f, 1f,
1f);
				//set transformation matrix for shader
				transform.set(camera.getCombined());
				transform.translate(obj.position.x, obj.position.y,
obj.position.z);
				shader.setUniformMatrix("u_projView", transform);
				Asset.bulletModel.render(shader.getShader());
			}
		shader.end();
		//set textLight shader is active to render player and invaders
later
		shader.setActiveShader("textLight");
	}
```

As you can see, to render all the bullets, you just call the begin () method of the shader only once. Limit number of call begin() method of shader will bring your game to a better performance.

Make the Enemy and EnemyCollection

to optimize game, the enemy can be built is similar bullet. But here i intend to show you how to use the DefaultObject3D class and DefaultObject3DCollection class, We will apply two class to build the Enemy and EnemyCollection class.

Example: `public class Enemy extends DefaultObject3D {/ / ... }`

Enemy, you can even completely customize the rendering stage model, simply you need to override method renderGL1 () for OpenGL ES 1.x or method renderGL2 () - OpenGL ES 2.0 for the class DefaultObject3D and get rid of unnecessary handling from parent class.

Example: Reduce render processing of Enemy overriden the process of rendering the DefaultObject3D.

```
@Override
public void renderGL2(float gameTime)
{
	texture.bind();
	shader.begin();
	setLightParameters();
	normal.idt();
	normal.rotate(0, 1, 0, rotate.y);
	normal3.set(normal.toNormalMatrix());
	shader.setUniformMatrix("u_normal", normal3);
	transform.set(camera.getCombined());
	transform.translate(position.x, position.y, position.z);
	transform.rotate(0, 1, 0, rotate.y);
	shader.setUniformMatrix("u_projView", transform);
	model.render(shader.getShader());
	shader.end();
}
```

objects DefaultObject3D also has its own collection class, that DefaultObject3DCollection. Simply after create new Default object 3D, you add the object to the instance of DefaultObject3DCollection instead add to scene directly. Of course then you still have to add the DefaultObject3DCollection to the game scene for make this collection can auto-update, auto-render …

But in spacewar3D game, the Enemy will be added directly into Game scene, so you don't see the class EnemyCollection.

Using Light Manager in GDX 3D game engine.

I can assure you that the light is always **Indispensable** when you develop 3D game. Because if there are no lights, then, do you distinguish a 3D space? You can not see the bright and dark regions on the object, the object is no drop shadow on the terrain or the object itself, the worse is the whole 3D space can only be a dark color. A 3D game no lights is not different from a 2D game.

When you use LightManager, you need ambientLight parameter, there is a small color value, is added to the final synthesis of color on each pixel in the Fragment Processor of the shader. If you have knowledge of [OpenGL ES shader language], you can see the file in light-tex-fs.glsl in the asset / data / shader folder ambientLight used to PREVENT your 3D Scene too dark Because somewhere on scene lights may not work.

Example: `LightManager lightManager = new LightManager(new Vector3 (0.1f, 0.1f, 0.1f));`

One problem is that how much light in a 3D space? There are a lot and how you can manage all of the lights, took out the light you need and remove the ones you do not need them? Meanwhile, the role of LightManager will be shown. You can create a new light, add light and LightManager and get light also from LightManager. LightManager is a 3D Services over it is always present in any 3D object. LightManager solve the problem of having a variety of types of lights in Scene. Light maybe Direction Light, PointLight, Spotlight … but all lights always using the same parent class, BaseLight class, you can cast BaseLight to Special Light class you need with method get (key, Class) of the LightManager.

Example: cast a light to DirectionLight

```
directlight = light.get("MainLight",DirectionLight.classes) ;// light is LightManager
```

Of course, To make get a light possible, you had to add the light before:

```
this.light.clear (); / / light is
```

when you get the light, you can use light to transmit values for the shader 's light parameters by overriding method setLightParameters () or pass a default object ISetLight to the 3D object.

```
	@Override
	public void setLightParameters() {

		shader.setUniformVector("u_light_ambient_color",
light.ambientColor);
		shader.setUniformVector("u_light_color",
directlight.getLightColor());
		shader.setUniformVector("u_light",
directlight.getLightDirection());
	}
```

spacewar3D, both Player and Enemy needs to light, and set values for lightParameters. So neat to program, we create class SpaceWarGameObject3D from the DefaultObject3D and override method setLightParameters () of DefaultObject3D in the class SpaceWarGameObject3D.

When Player spacewar3D the take damage, mainLight will turn red for 0.5 seconds. , This is very easily done by set the color in reference variable *mainLight* in the Player class. mainLight direct reference direction light in the game so it will influence the light that shine on the whole objects of the enemy and the player object.

However, handle light with shaders in game3D always is hard work for the GPU. Especially when the game has many different light sources. Thus let 's reduce the amount of lights you must use at a minimum. Also you can throw away the light feature for the object is not so Important, for example the bullets are lighted unnecessary, you only need to render the model of the bullet up and Determine to give it a color. Shader is only supported in OpenGL ES 2.0, If the device does not support OpenGL version, to create light, you must use the method that OpenGL ES 1.x has built for you. You can not make the option by shader.

Use the ShaderManager Control different render process of objectes

Above the mentioned bullet I will render without lighting, mà Means the process of rendering of the bullet will be different from the Player or Enemy so bullets will use

different shader to render. Sometime in the 3D games have usage of different shaders to render different object groups!

ShaderManager used to manage all the shader in your game. similar lightManager, you also add shader, get shader, set active a shader, set the value for shader parameters, call the method begin () of the shader …

Example: Render Bullets in BulletCollection use a different shader rendering Player or Enemy

```
	@Override
	public void renderGL2(float gameTime)
	{
		//set color shader is active to render bullets
		//shader is instance of ShaderManager
		shader.setActiveShader("color");
		//call begin() method here to reduce number of the method
calling that lead to better performance
		shader.begin();
		//render all bullets if the bullet is not dead
		for(Bullet obj : getObjectCollection())
			if(obj.isVisible())
			{
				if(obj.isBulletOfPlayer)
					shader.setUniformf("u_color", 1f, 1f, 0.5f,
1f);
				else
					shader.setUniformf("u_color", 0.5f, 1f, 1f,
1f);
				//set transformation matrix for shader
				transform.set(camera.getCombined());
				transform.translate(obj.position.x, obj.position.y,
obj.position.z);
				shader.setUniformMatrix("u_projView", transform);
				Asset.bulletModel.render(shader.getShader());
			}
		shader.end();
		//set textLight shader is default active shader to render
player and invaders later
		shader.setActiveShader("textLight");
	}
```

Here, to render all bullets, we have used shader "color", specified in method **setActiveShader**(). then you need to call begin () method of shader. You transmit values for shader parameters. then you render the model of bullet by method getShader () that will

retrieve the current active shader. The last, you call method end() of the shader and set active shaders is "textLight”- It is default shader used when you render the player or enemy in the game scene.

Using CameraManager control many Cameras in the game

Like light, I can assure you that the camera is always **Indispensable** when you develop 3D game. Cameras like the "eyes" of the players, observed every object in your 3D space. Of course a 3D game will have at least one or more cameras. example of a role-playing game, you can have ThirdPersonShooter (TPS) Camera that follows Player, or FixedCamera to the observer a fixed point in the game … Camera Manager will manage all in-game cameras. Camera is very necessary because there is a critical parameter when you render the 3D object, which is combined Matrix. This Matrix is combination of the Matrix View and Projection Matrix of a camera3D. Combined Matrix used as the genesis of the 3D transformation matrix of the object, and the shader using the transformation matrix to render objects:

For example:

```
//set combined matrix for the transformation matrix from CameraManager
transform.set(camera.getCombined ());
/ / Do more stuffs for transformation matrix like rotate, scale, ...
/ / set the transformation matrix for shader
shader. setUniformMatrix("u_projView", transform);
```

You can add the camera, get the camera and set the active camera using setActive…() method like the shader manager and light manager. Method getCombined () Camera Manager is get the Combined matrix of the active camera. You could refer source code of the engine for more information.

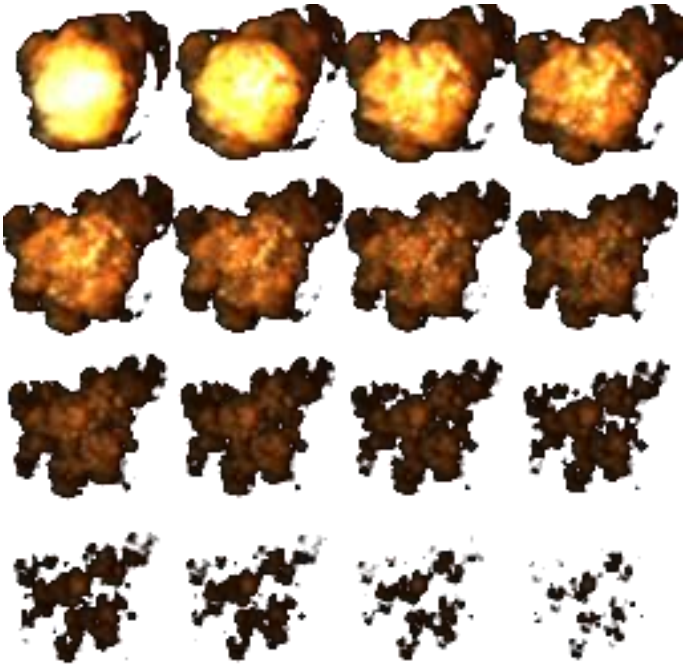
cameras need to be updated constantly, often a camera will be using the update () method by default, but these special cases, you will have to write using the update () method. example in spacewar3D , we use TPS Camera that follow **player** object, so we will need to update the TPS camera in the **Player** class

Example: activeCamera.Update (rotate.y, up, position, gameTime); / / activeCamera is TPS Camera

In BaseGameScene3D class, method update () method of the camera is not called, so you need to call update () to the camera in the sub-class of BaseGameScene3D to update camera.

Render an animated 2D texture on 3D

When there is a collision between objects in spacewar3D , usually there will be an explosion to illustrate the collision. During game2D spacewar, we use the particle effect, but the current particle effect can only be used for 2D games. Besides particle effects are consuming so much resources and make lower performance. Consequently we can use animated texture to represent an explosion occurred, as shown below:



If you already have the basic concepts of 3D graphics programming way, you will easily understand rendering techniques. 2D image here above has been split out and draw a square up from the pairs of vertices, is what has created in variable *Mesh* of game asset. 4 vertices will pair up to render a square contained texture from file explode.png

example: square Render:

```
Asset.explosionMesh.render(shader.getShader (), GL10.GL_TRIANGLE_FAN,
            (int)(obj.aliveTime / Explosion.EXPLOSION_LIVE_TIME * 15) * 4,
4);
```

The code to render explosions located in the ExplosionCollection.java file

Example: Render all explosion

```
Gdx.gl.glEnable(GL10.GL_BLEND); //Enable Alpha blend to
support transparent

Gdx.gl.glBlendFunc(GL10.GL_SRC_ALPHA,
GL10.GL_ONE_MINUS_SRC_ALPHA);

Asset.explosionTexture.bind();
shader.setActiveShader("text");
shader.begin();
shader.setUniformi("u_diffuse", 0);
transform .set(camera.getCombined());
for(Explosion obj : objectCollection)
     if(obj.isVisible())
     {

      transform.translate(obj.position.x, obj.position.y,
obj.position.z);
      shader.setUniformMatrix("u_projView", transform);
      Asset.explosionMesh.render(shader.getShader(),
GL10.GL_TRIANGLE_FAN,(int)(obj.aliveTime / Explosion.EXPLOSION_LIVE_TIME
* 15) * 4, 4);
     }

shader.end();
Gdx.gl.glDisable(GL10.GL_BLEND);
shader.setActiveShader("textLight");
```

As you can see in this code using the "text" shader to render using texture. Unlike the "color" shader which used to render bullet, the “text” shader supports texture-based render while “color” shader only supports color-based render. After the render is done, you set active shaders is "textLight” shader - It is default shader which used when you render the player or enemy in the game scene.

Gdx Engine Packages

Introduce to Gdx Engine Packages

This tutorial will describe lightly about packages of Gdx Engine.

1. Com.gdxengine.framework / Com.gdxengine.framework.interfaces

You can see almost base classes and interfaces in here. Many files in abstract classes và interfaces. With abstract class, they have prefix "Base" in class name to indicate that is abstract class. With interface, they have prefix 'I' in class name to indicate that is interface. Beside that, engine has built standard class inheritance and implementation from an abstract class and interface, we usually have prefix 'Default' before the name to the class. You can create a new class by inheriting from the 'Default' class (the fastest way) or from the abstract class and implements other interfaces, if necessary optimize the class.

1. Com.gdxengine.camera

Contains the camera is built **exclusively for the game 3D.** Camera should be added to the CameraManager Object to use

2. Com.gdxengine.effect

contains the effect that GDX Engine was built. Particle Effect Manager used to create particle effects **only for 2D games.** Addition class ShaderManager will manage shaders, to render 3D objects, **just for 3D gaming.**

3. Com.gdxengine.material

not built much. Material mainly used to store the values of parameters passed to the shader. These are the parameters used for the texture when rendering objects. These parameters may be the model textures, TextureSampler, NormalTexture ...

4. Com.gdxengine.light

No built many. Light mainly used to store the values of parameters passed to the shader. These are the parameters used for the light to render objects. These parameters can be type LightColor, LightDirection, LightPosition ...

5. Com.gdxengine.object3d

Contains the standard extension classes based on the parent class in the package framework or framework.Interfaces. You can rely on the classes in this package to build 3D objects for your own 3D games

6. Com.gdxengine.ui

Contains the class supports the creation of user interface components in the game. I have built the class to create a custom menu. Other components such as dialog, input text, checkbox, ... will continue to be added in the update of GDX engine.

7. Com.gdxengine.test. *

Contains classes show some individual features of GDX Engine or contains sub package contains the source code of the example game is available that GDX Engine has provided.