minigameslib.de

MinigamesLib

Minigames Library – A library to write minigames

mepeisen
30.01.2017

1 Table of contents

2 Preamble

MinigamesLib was originally created by InstanceLabs. It was meant as a library to support some of the Minecraft Minigames he created for Bukkit. You can view his profile at github: https://github.com/instance01

However, he decided to stop developing and supporting his work. As users, we felt sad about this and decided to ask him for overtaking the project. He agreed and so MysticCity became new project owner. I became main developer and contributor after some other guys developed only for some weeks but left the project. ***My nick name is mepeisen***.

While doing bug fixes and migration to newer Minecraft versions we decided to make a huge redesign and rework. We continued the work for version 2.0 of MinigamesLib. After doing some of the work we came up with the idea to completely rewrite MinigamesApi from scratch.

The result are two plugins:

- McLib (***M***inigames ***C***ore ***Lib***rary)
- MinigamesLib (The library to write Minigames)

Special thanks to all the users supporting the work. Every single review and feature wish was respected and will be part of the minigames system in version 2.0

3 License and usage scenarios

The library is built of multiple parts. We have client parts, support for testing and server libraries. You should read the following chapters carefully and decide which scenario fits your needs.

3.1 License

MinigamesLib contains a public part. It can always be used with GPLv3 in mind.

To fund the project, we decided to provide a commercial license. This is required because the commercial license may give more advanced warranty but does some restrictions to not break the warranty. The second cause are other commercial projects we might support. They are not happy with GPLs copy-left.

MinigamesLib itself will support some premium options that are covered by the commercial license too. More details will be part of future versions.

You are always allowed to fork the project under terms of GPL, specially while you always publish the source code. If you do not follow this license you are not allowed to modify the code in any way. But it is cleverer to ask us for feature requests directly or to contribute your work back to us. Contributing will guarantee that your feature is staying in the core lib and will be migrated to newer Minecraft versions from us.

3.2 Productional Server (Spigot)

It is fairly simple. The MinigamesLib is a plugin itself. Simply download the server jar from your favorite location and store the jar file into the plugins folder of your spigot server. Ensure you have a compatible version of McLib installed in plugins folder.

You need no special setup to work with MinigamesLib. It runs “out-of-the-box”. Some of the features may require additional setup to perform in a way you want them but this is mentioned in a detailed explanation of the features later on.

3.3 Developing (Eclipse-Setup)

We always wanted to have a smart IDE support. I decided to use eclipse for developing the minigames.

Read the McLib manual for detailed explanation of the minigames eclipse plugins.

3.4 Client (Forge)

MinigamesLib has its own client mod. It is optional. You may use it for your work or simply ignore it. The MinigamesLib will always work without the forge mod. But there are some nice features we get with our forge mod.

Read the McLib manual for detailed explanation of the minigames forge mod.

The premium part of our plugin will require the forge mod for some of the aspects.

3.5 Alternative client mods

We know that downloading and installing additional software on your PC and even on PC of your users is critical because hackers might introduce Trojan and some kind of virus. Some users do not like it and as long as MInigamesLib version 2 is new, users may be prejudiced.

MinigamesLib may support other clients in future versions as an alternative so that users will trust the MinigamesLib. However, the communication protocol itself and the MingamesLib client mod will stay open sourced to increase trust.

4 Development setup

We assume that you already read the McLib manual and have a working setup for it. We assume that you already created your new project as a starting point for your plugin.

Now add the following dependency to your pom.xml:

```
<dependency>
    <groupId>de.minigameslib.mglib</groupId>
    <artifactId>MinigamsAPI</artifactId>
    <version>2.0.0-SNAPSHOT</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>de.minigameslib.mglib</groupId>
    <artifactId>MinigamsPlugin</artifactId>
    <version>2.0.0-SNAPSHOT</version>
    <scope>runtime</scope>
</dependency>
```

Within your plugin.yml add the following dependency:

```
depend: [MinigamesLib]
```

5 First steps

5.1 onEnable

Within your onEnable you should first check for the valid version.

```
if (MinigamesLibInterface.instance().getApiVersion() >=
    MinigamesLibInterface.APIVERSION_2_0_0)
    throw new IllegalStateException("Incompatible MinigamesLib");
```

The version check is done via integers. See the Javadoc for explanation of the integers. In the code above we will support any 1.x api version.

If you require special features for a specific Minecraft version, you can add a second version check. Notice: It is only safe to check for the Minecraft version if the McLib version supports this Minecraft version!

```
if (MclibInterface.instance().getMinecraftVersion().isBelow(
    MinecraftVersionsType.V1_9)
    throw new IllegalStateException("require at least 1.9");
```

After version checks you should register your localized messages and other enumerations we will need in minigames code later on. See McLib manual for details on enumeration handling.

```
EnumServiceInterface.instance().registerEnumClass(this, MyMessages.class);
EnumServiceInterface.instance().registerEnumClass(this, MyArenaTypes.class);
```

After initialization of enumerations we will add the minigame itself.

```
try
{
    MinigamesLibInterface.instance().initMinigame(this, new MyProvider());
}
catch (McException e)
{
    throw new IllegalStateException("Init failure", e);
}
```

5.2 onDisable

Within your onDisable simply add the following line:

```
EnumServiceInterface.instance().unregisterAllEnumerations(this);
```

5.3 Implementing MinigameProvider

Adding the minigame itself requires you to create a new class "MyProvider" implementing the MinigameProvider interface.

This will introduce some methods mainly used for information purpose. You should add messages to MyMessage class and simply return them in the given methods (getShortDescription, getDescription etc.). See the Javadoc for details-

The most important method is getName(). It returns the technical name of the minigame. It is recommnded to return the plugin name in this method.

5.4 MyArenaTypes

This type is an enumeration for your arena types. Many minigames do only support a single arena type. Thus, your enumeration will only have one entry:

```
public enum MyArenaTypes implements ArenaTypeInterface
{
    @ArenaType(MyStandardArena.class)
    Standard
}
```

When do you need multiple arena types?

Mainly there are two main arena types: A single player arena and a team arena. If you support both you will actually really need two arena types.

```
public enum MyArenaTypes implements ArenaTypeInterface
{
    @ArenaType(MyClassicArena.class)
    Standard,

    @ArenaType(MyTeamArena.class)
    Team
}
```

We do not expect that you need more arena types for other scenarios. If you want the administrators to influence the behavior of your arena match you can use the rules and their configuration (we will explain it later on).

6 My first game (Classic PvP game)

We will now learn how to implement your own game. We assume the following:

- We have a classic pvp arena (no teams): MyClassicArena
    - A player that dies will lose a life. After losing 5 lives the player will lose the whole game.
    - Last man standing wins
    - Each match is limited to 5 minutes. If time expires the remaining players win and take places in order of their kill statistics
- We have a team arena (teams against other teams): MyTeamArena
    - A player that dies will lose a life. After losing 5 lives the player will be leaving the game but he does not yet lose.
    - If all players of a team lost their lives all team members lose.
    - Last team having a player with lives wins
    - Each match is limited to 5 minutes. If time expires the remaining team wins and takes places in order of their kill statistics.
- Each team member gets an unbreakable sword at match startup
- Blocks cannot be broken, simply fight
- Leaving the arena means to leave the game
- There is a “hospital zone”, healing wounded players but only once per 20 seconds.

6.1 Theory behind Minigames

Before you start with your minigame let me explain some things.

In the following chapters, everything is done in “Rules”. You may wonder what a rule is. From game theory, a rule is some kind of single method for interaction with the game state. Multiple rules describe the game mechanics. You may read Wikipedia at https://en.wikipedia.org/wiki/Game_mechanics for more detailed explanation.

Within MinigamesLib a rule is a single object fetching events from your arena and doing important things. Look at the bullet points above. Each bullet point will describe a rule. At least we are having the following rules:

1. Single Player: Player life management
2. Single Player: Last man standing wins
3. Single Player: Time Limit
4. Team Play: Player life management
5. Team Play: Team life management
6. Team Play: Last team standing wins
7. Team Play: Time limit
8. Give unbreakable sword at begin
9. No Block breaks
10. Leaving arena
11. Heal in hospital

In the following chapters, we will implement the rules step by step. However, the rules always have a focus. Most rules are focused on the game itself. But rules 9 to 11 are focused on zones. If you do not know what a zone is you should really read the McLib manual.

Actually, we have more rules under the hood. They are not directly visible. For example, there is a rule to port players to their spawns… I will come back to it later on.

6.2 Create and register your rules

First let us create an enumeration.

```
public enum MyArenaRules implements ArenaRuleSetType
{
    SinglePlayerLifeManagement,
    SingleLastManStandingWins,
    SingleTimeLimit,
    TeamsPlayerLifeManagement,
    TeamsTeamLifeManagement,
    TeamsLastTeamStandingWins,
    TeamsTimeLimit,
    GiveSword
}
public enum MyZoneRules implements ZoneRuleSetType
{
    NoBlockBreaks,
```

```
    LoseOnLeavingArena,
    HealInHospital
}
```

As always, we register this enumeration in onEnable:

```
EnumServicesInterface.instance().registerEnumClass(this, MyArenaRules.class);
EnumServicesInterface.instance().registerEnumClass(this, MyZoneRules.class);
```

We now need to tell the MinigamesLib where to find the implementation of our rules. This is done in onEnable after registering the enumeration itself:

```
MinigamesLibInterface.instance().registerRuleSet(
    this,
    MyArenaRules.SinglePlayerLifeManagementRule.class,
    SinglePlayerLifeManagementRule::new);
```

Do this for every rule set. You need to create the rule set classes itself:

```
public class SinglePlayerLifeManagement extends AbstractArenaRule
{
    public SinglePlayerLifeManagement(
        ArenaRuleSetType type, ArenaInterface arena) throws McException
    {
        super(type, arena);
        // do some initialization
        // you may throw an exception if the arena setup or config is invalid
    }
}
```

6.3 Implementing MyClassicArena

Create the class MyClassicArena and extend “ClassicSinglePlayerArena”. This base class already does many important things. We will extend the arena setup in the following steps.

As you can see you will be forced to implement some informational methods: getDisplayName, getShortDescription, getDescription. Create some localized messages in MyMessages enumeration and return them in your arena class.

6.3.1 The fixed arena rules.

Arena rules are focused on the game or match itself. You already may have an idea that rules 1 to 3 and rule 8 are used as our arena rules for the single player match.

Fixed arena rules cannot be removed by administrators. The game requires them for the main game mechanics. Later on, we will give some explanation on optional rules.

Now let us override and influence getFixedArenaRules in MyClassicArena.

```
@Override
public Set<ArenaRuleSetType> getFixedArenaRules()
{
    final Set<ArenaRuleSetType> result = super.getFixedArenaRules();
```

```
        result.add(MyArenaRules.SinglePlayerLifeManagement);
        result.add(MyArenaRules.SingleLastManStandingWins);
        result.add(MyArenaRules.SingleTimeLimit);
        result.add(MyArenaRules.GiveSword);
        return result;
    }
```

For the moment, we do not need any additional setup.

6.3.2 SinglePlayerLifeManagement

Let us divide this rule set into the following sub pieces:

- At beginning set the lives for each player
- Decrement the lives on death
- Reaching 0 lives let the player lose the game.

Before starting let us first find a way to store useful information. We use the game statistic. A game statistic is an integer stored for teams or for single players. It is perfect to store the lives of players there.

Create an enumeration and register it on your onEnable method.

```
public enum MyStatistics implements MatchStatiscticId
{
    PlayerLives
}

EnumServicesInterface.instance().registerEnumClass(this, MyStatistics.class);
```

As I already told each rule is about fetching events and doing some action. The match begins by fetching the arena state.

```
@McEventHandler
public void onMatchStart(ArenaStateChangedEvent evt)
{
    if (evt.getNewState() == ArenaState.PreMatch)
    {
        final ArenaMatchInterface match = evt.getArena().getCurrentMatch();
        match.getPlayers().forEach(
            p -> match.setStatistic(p, MyStatistics.PlayerLives, 5));
    }
}
```

What do we do here? We are reacting on the arena state „PreMatch“. This state is set as soon as the match is starting. To make it more clear: This state is set right before the match cooldown starts. We are receiving the current match and the players within the match.

We set the PlayerLives statistic to a hard-coded value (5). We do not focus on how to make the value changeable by admins. This is covered in chapter 7.2 Rule set configuration.

Now let us decrement the player statistic on death and handle the player lose.

```
@McEventHandler
public void onDeath(ArenaPlayerDiesEvent evt)
{
    if (evt.getPlayer().isPlaying())
    {
        final ArenaMatchInterface match = evt.getArena().getCurrentMatch();
        if (match.decStatistic(
            evt.getPlayer().getPlayerUUID(),
            MyStatistics.PlayerLives,
            1) <= 0)
        {
            evt.getPlayer().lose();
        }
    }
}
```

We first check if the player of this event is actually playing within the game. Maybe we got a spectator jumping into some lava or similar things. We do not want to catch spectator events.

After checking for spectators, we decrement the player lives by 1. The method “decStatistic” returns the new value and we check it for being less or equal to zero. Can the player lives be negative? This should never happen in our game mechanics but maybe there is another rule or plugin influencing our statistics from outside. It is more robust to check for “negative” lives too.

That’s it. We implemented our single player live management.

6.3.3 SingleLastManStandingWins

NOTE: This rule is already builtin. You can use “BasicWinningRuleSets.LastManStanding”. It contains the same code. In this chapter I only want to explain the code itself.

What do we want to do for implementing this rule? It is fairly simple. Fetch every situation where a player leaves the game. Maybe because of connection losses or because he loses the game. Check if there is a last man standing (only one player left) and let this player win the game.

There is an event covering every situation we need because every time a player leaves before marked as winner he automatically loses the game:

```
@McEventHandler
public void onPlayerLeaves(ArenaPlayerLoseEvent evt)
{
    final ArenaMatchInterface match = evt.getArena().getCurrentMatch();
    if (match.getPlayerCount() == 1)
    {
        match.setWinner(match.getPlayers().iterator.next());
    }
}
```

Again, this is a fairly simple rule. We query the current player count (active players to yet marked as winners or losers). If there is only one player left, he wins the match.

6.3.4 SingleTimeLimit

TODO

7 Additional topics

7.1 Optional rules vs. fixed rules

TODO

7.2 Rule set configuration

TODO

7.3 Predefined rules

TODO

7.4 Places (1st Winner, 2nd winner etc.)

TODO