Per-Competition Code

Now that you have an Arena class with functioning game logic, it's time to expand on that.

Creating Competition Classes

As mentioned in the Adding Game Logic page, you will need to create a custom Competition class. The following code below is how to create a custom Competition:

```
public class MyCompetition extends LiveCompetition<MyCompetition> {

    public MyCompetition(MyArena arena, CompetitionType type, ArenaMap map) {
        super(arena, type, map);
    }

}
```

As seen above, you will need to extend the **LiveCompetition** class, which is created for competitions that are live on the server BattleArena is running on. On it's own, this will do nothing, so the next step is to create a custom **ArenaMap** which is responsible for creating the competition.

```
public class MyCompetitionMap extends LiveCompetitionMap {
    public static final MapFactory FACTORY = MapyFactory.create(MyCompetitionMap.class,
MyCompetitionMap::new);

    public MyCompetitionMap() {
    }

    public MyCompetitionMap(String name, Arena arena, MapType type, String world, @Nullable
Bounds bounds, @Nullable Spawns spawns) {
        super(name, arena, type, world, bounds, spawns);
    }

    // Override this method in order to use your custom competition class
    @Override
    public LiveCompetition<?> createCompetition(Arena arena) {
        if (!(arena instanceof MyArena myArena)) {
            throw new IllegalArgumentException("Arena must be an instance of MyArena!");
```

```
        }

        return new MyCompetition(myArena, arena.getType(), this);
    }
}
```

As mentioned in an earlier segment of this documentation, maps can exist without necessarily having a competition bound to them, meaning they are responsible for actually creating a live competition. In the map class above, it can be seen the **createCompetition** method is overridden to instead create an instance of our **MyCompetition** class.

Additionally, a **MapFactory** is specified at the top of the class - this is important for the next step of linking this to your **MyArena** so BattleArena knows which ArenaMap (and therefore, Competition) to create for your Arena. It is also **very important** that both constructors are specified as seen in the example above.

And finally, the last step is to override the **getMapFactory** method in **MyArena**, and specify your factory like so:

```
public class MyArena extends Arena {
    @ArenaOption(name = "infection-time", description = "How long a player should be infected once hit.")
    private Duration infectionTime = Duration.ofSeconds(5);

    private final Set<UUID> infectedPlayers = new HashSet<>();

    @Override
    public MapFactory getMapFactory() {
        return MyCompetitionMap.FACTORY;
    }

    ...
  }
```

Per-Competition Code

Now that we have all the necessary classes created, you can now start creating code that will exist on a per-competition level. If we wanted to infect a random player every minute for instance the following could be done like so:

```
public class MyCompetition extends LiveCompetition<MyCompetition> {
```

```
    private BukkitTask tickTask;

    public MyCompetition(MyArena arena, CompetitionType type, LiveCompetitionMap map) {
        super(arena, type, map);
    }

    public void startInfectTask() {
        this.tickTask = Bukkit.getScheduler().runTaskTimer(this.getArena().getPlugin(),
this::infectPlayer, 0, 60 * 60 * 20);
    }

    public void stopInfectTask() {
        if (this.tickTask != null) {
            this.tickTask.cancel();
        }

        this.tickTask = null;
    }

    private void infectPlayer() {
        MyArena arena = (MyArena) this.getArena();

        // Infect a random player
        List<ArenaPlayer> uninfectedPlayers = this.getPlayers().stream().filter(player ->
!arena.isInfected(player)).toList();
        if (uninfectedPlayers.isEmpty()) {
            return;
        }

        ArenaPlayer player = uninfectedPlayers.get((int) (Math.random() *
uninfectedPlayers.size()));
        arena.infect(player.getPlayer());
    }
}
```

And with those changes, we also need to update **MyArena** to actually call the start and stop methods. Here is what the updated **MyArena** class would look like:

```
public class MyArena extends Arena {
    private static final String INFECTED_METADATA = "infected";
```

```
    @ArenaOption(name = "infection-time", description = "How long a player should be infected
once hit.")
    private Duration infectionTime = Duration.ofSeconds(5);

    @Override
    public MapFactory getMapFactory() {
        return MyCompetitionMap.FACTORY;
    }

    @ArenaEventHandler
    public void onDamageEntity(EntityDamageByEntityEvent event) {
        if (event.getDamager() instanceof Player damager && event.getEntity() instanceof
Player player) {
            // Player is not infected, let's infect them :)
            if (!player.hasMetadata(INFECTED_METADATA)) {
                this.infect(player);

                damager.sendMessage("You have infected " + player.getName() + "!");
            }
        }
    }

    @ArenaEventHandler
    public void onMove(PlayerMoveEvent event) {
        if (event.getPlayer().hasMetadata(INFECTED_METADATA)) {
            event.getPlayer().sendMessage("You are infected! You cannot move!");
            event.setCancelled(true);
        }
    }

    @ArenaEventHandler
    public void onPhaseStart(ArenaPhaseStartEvent event, MyCompetition competition) {
        // Ensure we are ingame
        if (!CompetitionPhaseType.INGAME.equals(event.getPhase().getType())) {
            return;
        }

        competition.startInfectTask();
    }
```

```
    @ArenaEventHandler
    public void onPhaseComplete(ArenaPhaseCompleteEvent event, MyCompetition competition) {
        // Ensure we are ingame
        if (!CompetitionPhaseType.INGAME.equals(event.getPhase().getType())) {
            return;
        }

        competition.stopInfectTask();
    }

    public boolean isInfected(ArenaPlayer player) {
        return player.getPlayer().hasMetadata(INFECTED_METADATA);
    }

    public void infect(Player player) {
        player.sendMessage("You have been infected!");
        player.setMetadata(INFECTED_METADATA, new FixedMetadataValue(MyPlugin.getInstance(),
true));

        // Infect the player for the given duration
        Bukkit.getScheduler().runTaskLater(MyPlugin.getInstance(), () -> {
            player.removeMetadata(INFECTED_METADATA, MyPlugin.getInstance());
            player.sendMessage("You are no longer infected!");
        }, this.infectionTime.toMillis() / 50);
    }
}
```

As seen above, when the game enters the **in-game** phase, we start running our task in the active **MyCompetition** to infect a random player every minute. Once the competition is no longer ingame, we stop the task.

Revision #4
Created 2024-12-14 14:04:29 UTC by Redned
Updated 2024-12-14 14:53:46 UTC by Redned