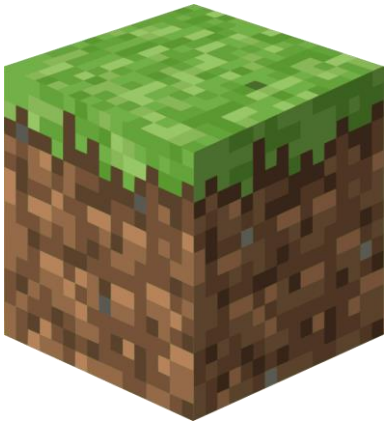
NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

Software Engineering Report

1st Semester 2024/2025

Minecraft World Edit



João Rivera	62877
Dimitrios Schoinas	65313
João Nascimento	62896
Diogo Mateus	65379
Gonçalo Cascais	60046

P R O J E C T M A N A G E M E N T

Scrum Master Identification

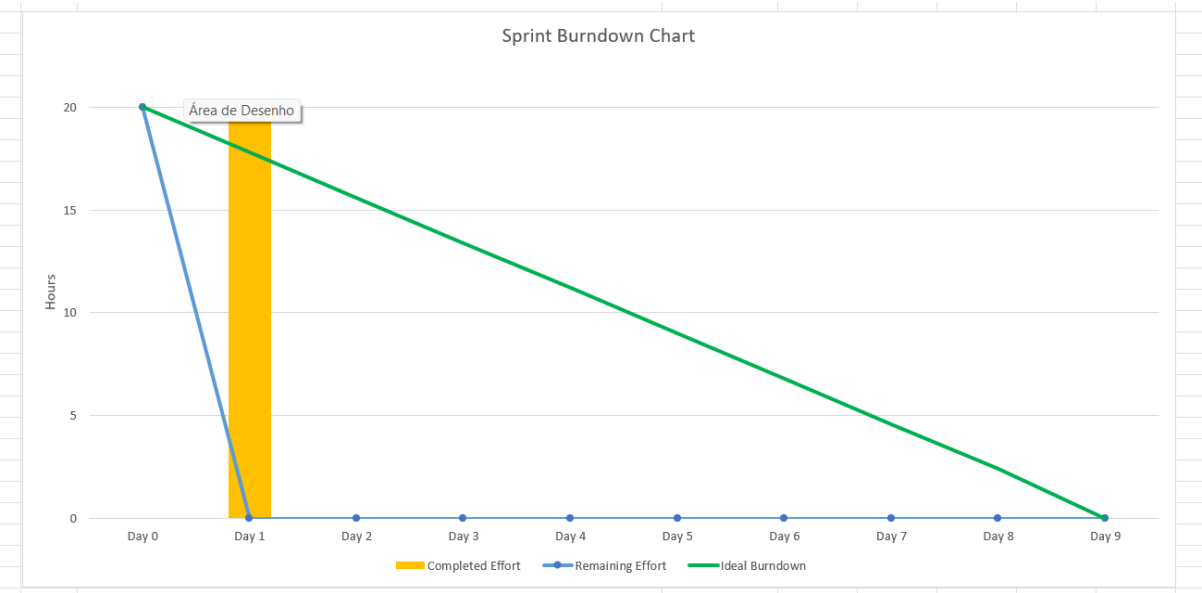
Week 1	João Rivera (62877)
Week 2	Dimitrios Schoinas (65313)
Week 3	Gonçalo Cascais (60046)
Week 4	João Nascimento (62896)
Week 5	Diogo Mateus (65379)
Week 6	João Rivera (62877)
Week 7	Dimitrios Schoinas (65313)
Week 8	Gonçalo Cascais (60046)

Burndown Chart and Scrum Board Evolution

(one chart and board per week)

WEEK 1 :

Burndown chart:



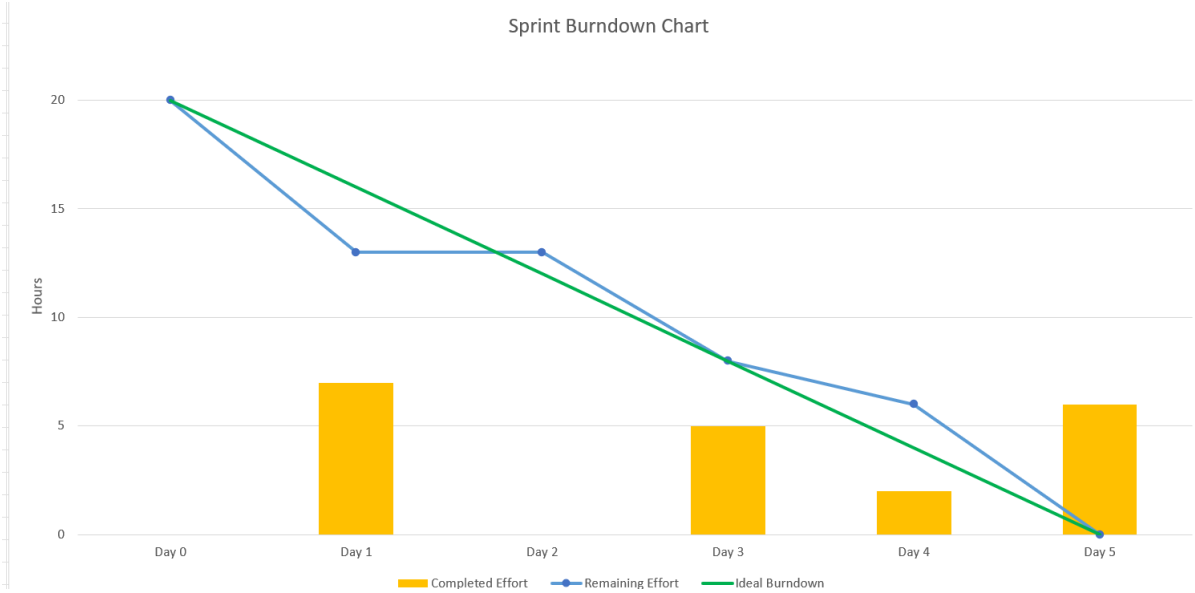
Scrum board:

SPRINT TASKS

Backlog	To Do	Doing	Reviewing	Done
				Read the project - everyone
				Installed the required files - everyone

WEEK 2

Burndown chart:



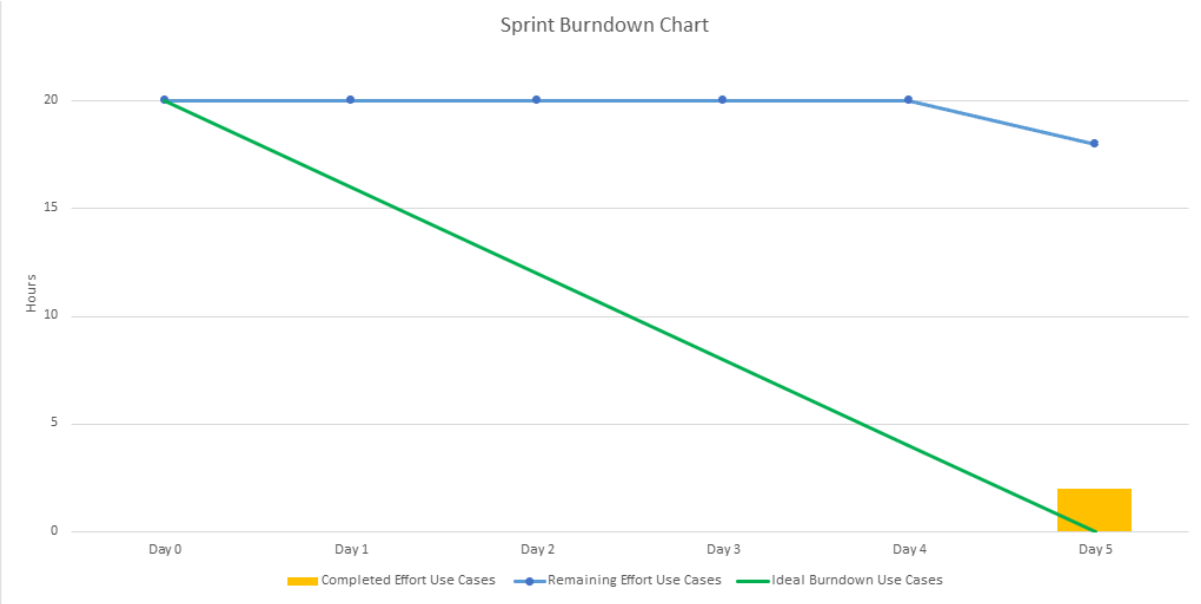
SCRUM BOARD:

SPRINT TASKS

Backlog	To Do	Doing	Reviewing	Done
User story 1 -As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.				Read the project - everyone
User story 2 - As a player I want a way to apply visual weather effects, such as rain, snow, and drought, to specific areas of the building. These effects would simulate the impact of weather on the blocks, altering their appearance to reflect wet, frozen or weathered conditions so that I can create a visually immersive experience.				Installed the required files - everyone
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.				

WEEK 3

Burndown chart:

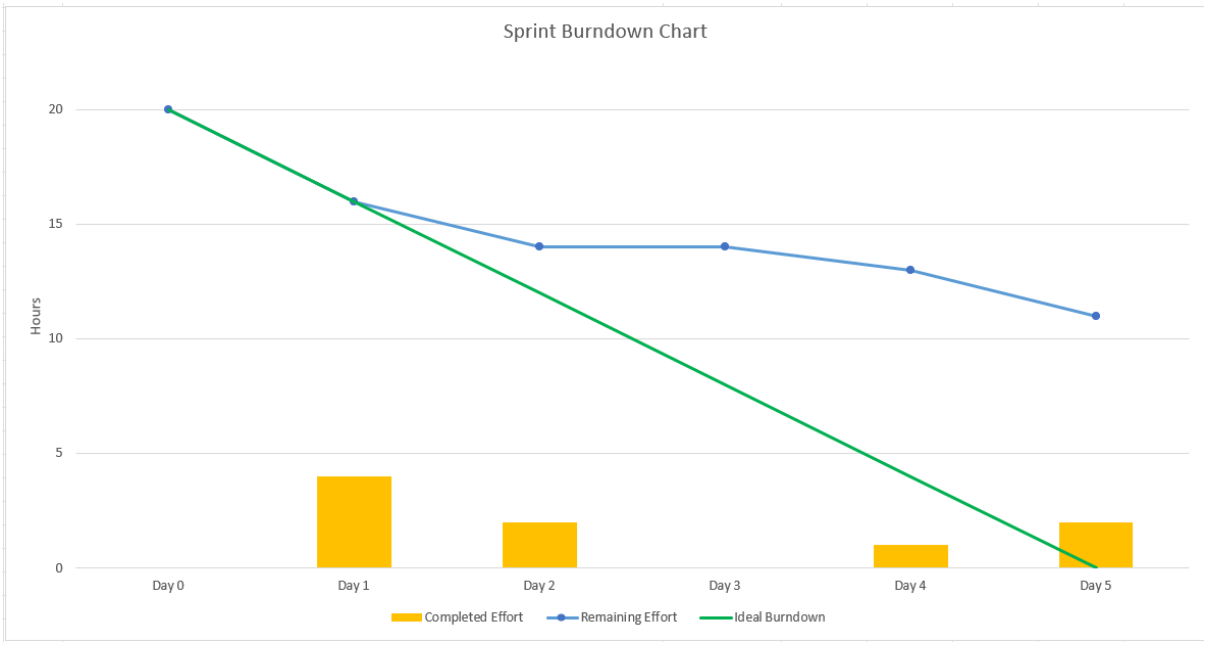


SCRUM BOARD:

Backlog	To Do	Doing	Reviewing	Done
User story 1 -As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.				Read the project - everyone
User story 2 - As a player I want a way to apply visual weather effects, such as rain, snow, and drought, to specific areas of the building. These effects would simulate the impact of weather on the blocks, altering their appearance to reflect wet, frozen or weathered conditions so that I can create a visually immersive experience.				Installed the required files - everyone
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.				
			Design patterns dimitri - (pessoas a dar review)	
		Code Metrics - Dimitri		
			Code Smells Dimitri - (pessoas a dar review)	
	Use Case - Dimitri			
	Design Patterns - Gonçalo			
	Code Metrics - Gonçalo			
	Code Smells - Gonçalo			
	Use Case - Gonçalo			
		Design Patterns - João Rivera		
	Code Metrics - João Rivera			
	Code Smells - João Rivera			
	Use Cases - João Rivera			
	Design Patterns - João Nascimento			
	Code Metrics - João Nascimento			
	Code Smells - João Nascimento			
	Use Cases - João Nascimento			
	Design Patterns - Diogo			
	Code Metrics - Diogo			
	Code Smells - Diogo			
	Use Cases - Diogo			

WEEK 4

Burndown chart:



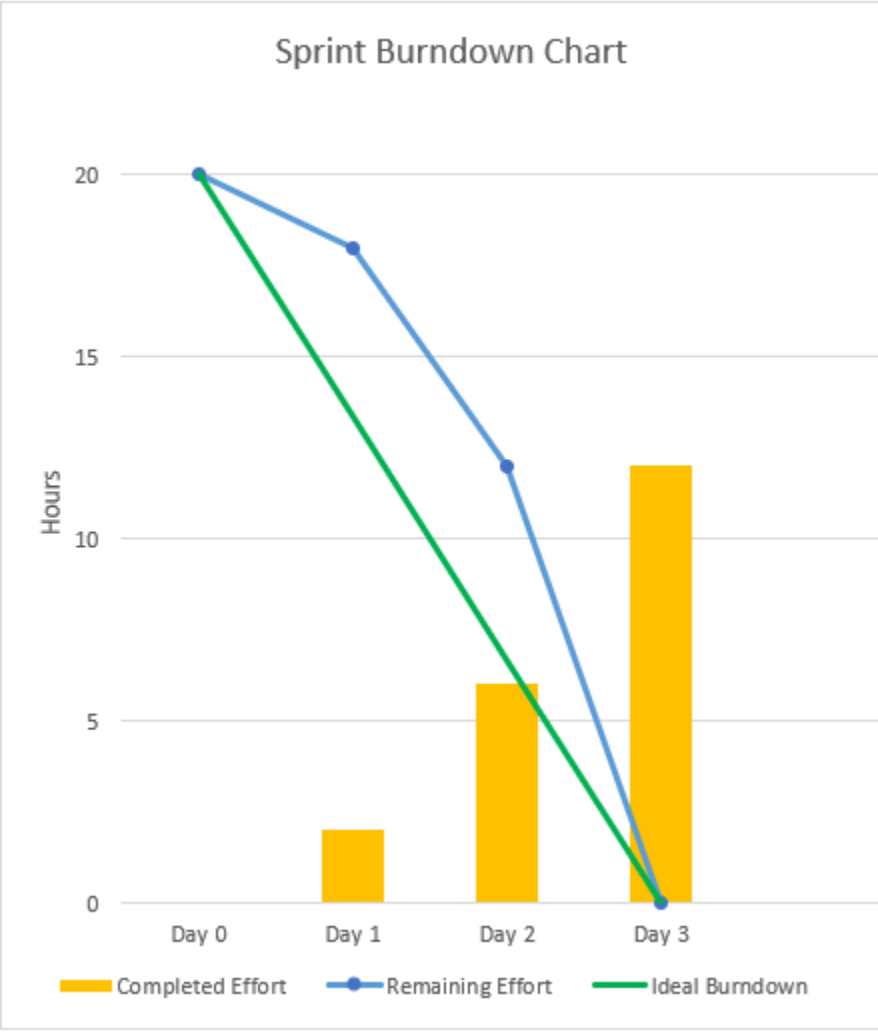
SCRUM BOARD:

SPRINT TASKS

Backlog	To Do	Doing	Reviewing	Done
User story 1 -As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.				Read the project - everyone
User story 2 - As a player I want a way to apply visual weather effects, such as rain, snow, and drought, to specific areas of the building. These effects would simulate the impact of weather on the blocks, altering their appearance to reflect wet, frozen or weathered conditions so that I can create a visually immersive experience.				Installed the required files - everyone
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.				
			Design patterns dimitri - (pessoas a dar review)	
			Code Metrics - Dimitri	
			Code Smells Dimitri - (pessoas a dar review)	
			Use Case - Dimitri	
			Design Patterns - Gonçalo	
	Code Metrics - Gonçalo			
			Code Smells - Gonçalo	
	Use Case - Gonçalo			
		Design Patterns - João Rivera		
		Code Metrics - João Rivera		
				Code Smells - João Rivera
		Use Cases - João Rivera		
		Design Patterns - João Nascimento		
	Code Metrics - João Nascimento			
			Code Smells - João Nascimento	
	Use Cases - João Nascimento			
			Design Patterns - Diogo	
	Code Metrics - Diogo			
				Code Smells - Diogo
		Use Cases - Diogo		

WEEK 5

Burndwon chart:


Sprint Burndown Chart
20
15
10
5
0
Hours
Day 0
Day 1
Day 2
Day 3
Completed Effort
Remaining Effort
Ideal Burndown

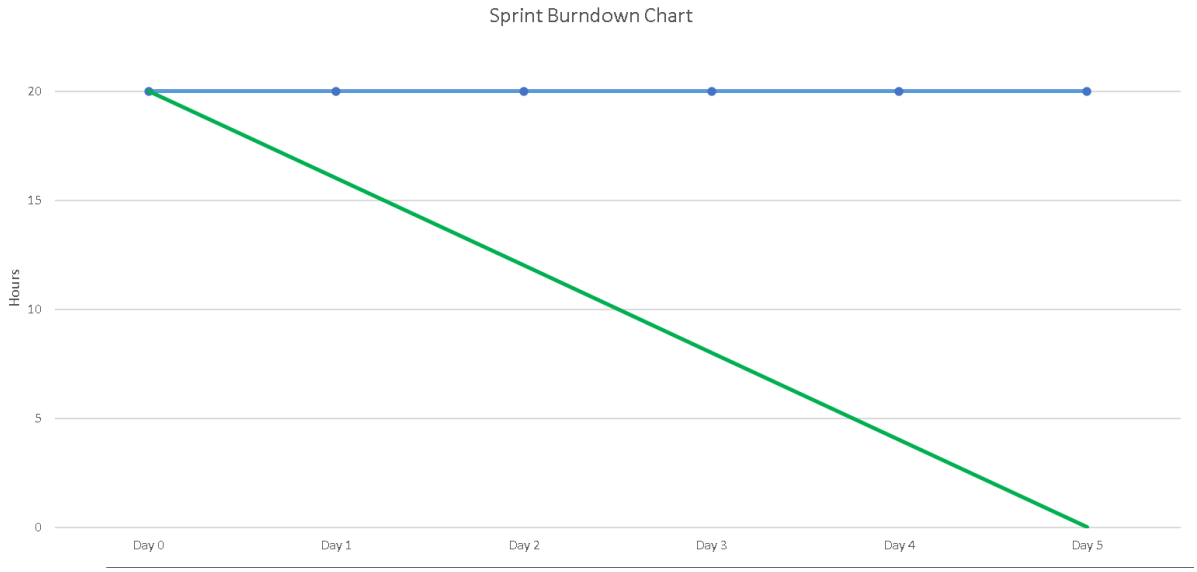
Scrum Board:

SPRINT TASKS

Backlog	To Do	Doing	Reviewing	Done
User story 1 -As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.				Read the project - everyone
User story 2 - As a player I want a way to apply visual weather effects, such as rain, snow, and drought, to specific areas of the building. These effects would simulate the impact of weather on the blocks, altering their appearance to reflect wet, frozen or weathered conditions so that I can create a visually immersive experience.				Installed the required files - everyone
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.				
				Design patterns dimitri
				Code Metrics - Dimitri
				Code Smells Dimitri
				Use Case - Dimitri
				Design Patterns - Gonçalo
				Code Metrics - Gonçalo
				Code Smells - Gonçalo
				Use Case - Gonçalo
				Design Patterns - João Rivera
				Code Metrics - João Rivera
				Code Smells - João Rivera
				Use Cases - João Rivera
				Design Patterns - João Nascimento
				Code Metrics - João Nascimento
				Code Smells - João Nascimento
				Use Cases - João Nascimento
				Design Patterns - Diogo
				Code Metrics - Diogo
				Code Smells - Diogo
				Use Cases - Diogo

WEEK 6

Burndown chart:

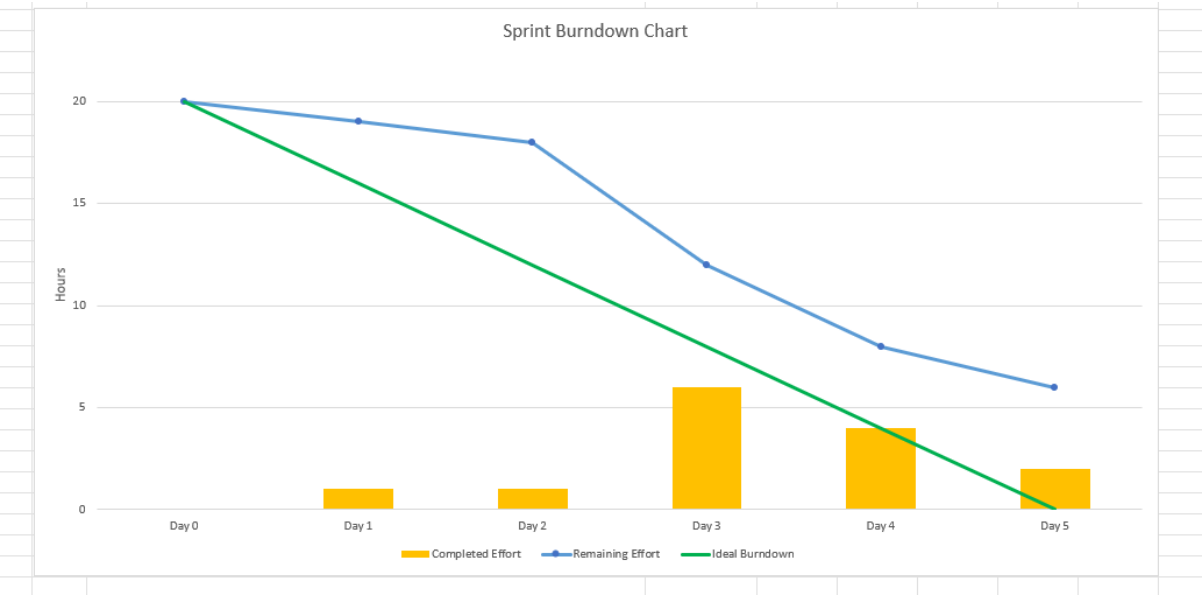


Scrum Board:

User story 1 -As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.	US3-T1 - Do the base of the lake command	US2-T2-Do the types "snow" and "drought" of the weather command - Dimitrios Schoinas		Read the project - everyone
ser story 2 - As a player I want a way to pply visual weather effects, such as rain, now, and drought, to specific areas of the uilding. These effects would simulate the npact of weather on the blocks, altering heir appearance to reflect wet, frozen or veathered conditions so that I can create visually immersive experience.	US3-T2 - Do the type "water" of the lake command			Installed the required files - everyone
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.	US3-T3 - Do the types "lava", "oasis", "pond" of the lake command			
	US1-T1- Do the clipboard command			Design patterns dimitri
	US2-T1-Do the types "hell" and "rain" of the weather command			Code Metrics - Dimitri
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.	US3-T3 - Do the types "lava", "oasis", "pond" of the lake command			
	US1-T1- Do the clipboard command			Design patterns dimitri
				Code Metrics - Dimitri
				Code Smells Dimitri
				Use Case - Dimitri
				Design Patterns - Gonçalo
				Code Metrics - Gonçalo
				Code Smells - Gonçalo
				Use Case - Gonçalo
				Design Patterns - João Rivera
				Code Metrics - João Rivera
				Code Smells - João Rivera
				Use Cases - João Rivera
				Design Patterns - João Nascimento
				Code Metrics - João Nascimento
				Code Smells - João Nascimento
				Use Cases - João Nascimento
				Design Patterns - Diogo
				Code Metrics - Diogo

WEEK 7

Burndown chart:



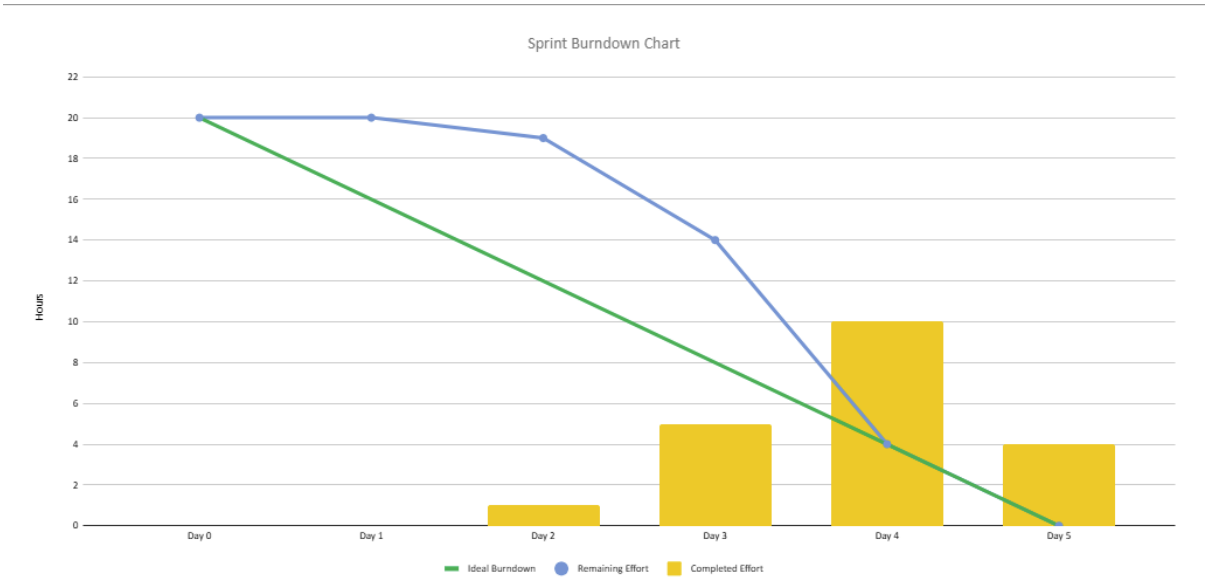
Scrum board:

SPRINT TASKS

Backlog	To Do	Doing	Reviewing	Done
User story 1 -As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.	Final Report	US3-T1 - Do the base of the lake command	Reviewing - Do the sub-commands "drought" and "snow" of the weather command - Reviewer: Diogo	US2-T1 - Do the sub-commands "drought" and "snow" of the weather command - (Dimitrios)
User story 2 - As a player I want a way to apply visual weather effects, such as rain, snow, drought and hell ,to specific areas of the building so that it is easier to change the aspect of my buildings depending on the weather.	Video	US3-T2 - Do the type "water" of the lake command		Read the project - everyone
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.	US1-T1- Do the clipboard command	US3-T3 - Do the types "lava", "oasis", "pond" of the lake command		Installed the required files - everyone
		US2-T2 - Do the sub-commands "hell" and "rain" of the weather command		Design patterns dimitri
				Code Metrics - Dimitri
				Code Smells Dimitri
				Use Case - Dimitri
				Design Patterns - Gonçalo
				Code Metrics - Gonçalo
				Code Smells - Gonçalo
				Use Case - Gonçalo
				Design Patterns - João Rivera
				Code Metrics - João Rivera
				Code Smells - João Rivera
				Use Cases - João Rivera
				Design Patterns - João Nascimento
				Code Metrics - João Nascimento
				Code Smells - João Nascimento
				Use Cases - João Nascimento
				Design Patterns - Diogo
				Code Metrics - Diogo
				Code Smells - Diogo
				Use Cases - Diogo

WEEK 8

Burndown chart:



Scrum board:

Backlog	To Do	Doing	Reviewing	Done
User story 1 -As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.				US2-T1 - Do the sub-commands "drought" and "snow" of the weather command - (Dimitrios)
User story 2 - As a player I want a way to apply visual weather effects, such as rain, snow, drought and hell ,to specific areas of the building so that it is easier to change the aspect of my buildings depending on the weather.				US3-T2 - Do the type "water" of the lake command -(Gonçalo)
User story 3- As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house giving it an amazing view and a feel of nature.				US3-T3 - Do the types "lava", "oasis", "pond" of the lake command - (João Rivera)
				US2-T2 - Do the sub-commands "hell" and "rain" of the weather command - (Diogo)
				US3-T1 - Do the base of the lake command - (Goncalo)
				US1-T1- Do the clipboard command - (João nascimento)
				Video
				Final Report
				Design patterns dimitri
				Read the project - everyone
				Installed the required files - everyone
				Code Metrics - Dimitri
				Code Smells Dimitri
				Use Case - Dimitri
				Design Patterns - Gonçalo
				Code Metrics - Gonçalo
				Code Smells - Gonçalo
				Use Case - Gonçalo
				Design Patterns - João Rivera
				Code Metrics - João Rivera
				Code Smells - João Rivera
				Use Cases - João Rivera
				Design Patterns - João Nascimento
				Code Metrics - João Nascimento
				Code Smells - João Nascimento
				Use Cases - João Nascimento
				Design Patterns - Diogo
				Code Metrics - Diogo
				Code Smells - Diogo
				Use Cases - Diogo

M I L E S T O N E 1

1. User Stories Description

1.1. [Clipboard feature] [João Nascimento 62896]

As a Minecraft Town Builder, I want a clipboard feature so that I can quickly save and organize multiple projects at once, helping me finish tasks more efficiently and aligning seamlessly with the overall mod experience.

1.2. [Dynamic Weather Effects on Builds] [Dimitrios Schoinas 65313]
As a player I want a way to apply visual weather effects, such as rain, snow, drought and hell ,to specific areas of the building so that it is easier to change the aspect of my buildings depending on the weather.
Note These effects would simulate the impact of weather on the blocks, altering their appearance to reflect wet, frozen or weathered conditions so that I can create a visually immersive experience.

1.3. [Artificial Lake creator] [Gonçalo Cascais 60046]
As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house so that I can give it an amazing view and a feel of nature.

M I L E S T O N E 2

1. Non-trivial Design Patterns

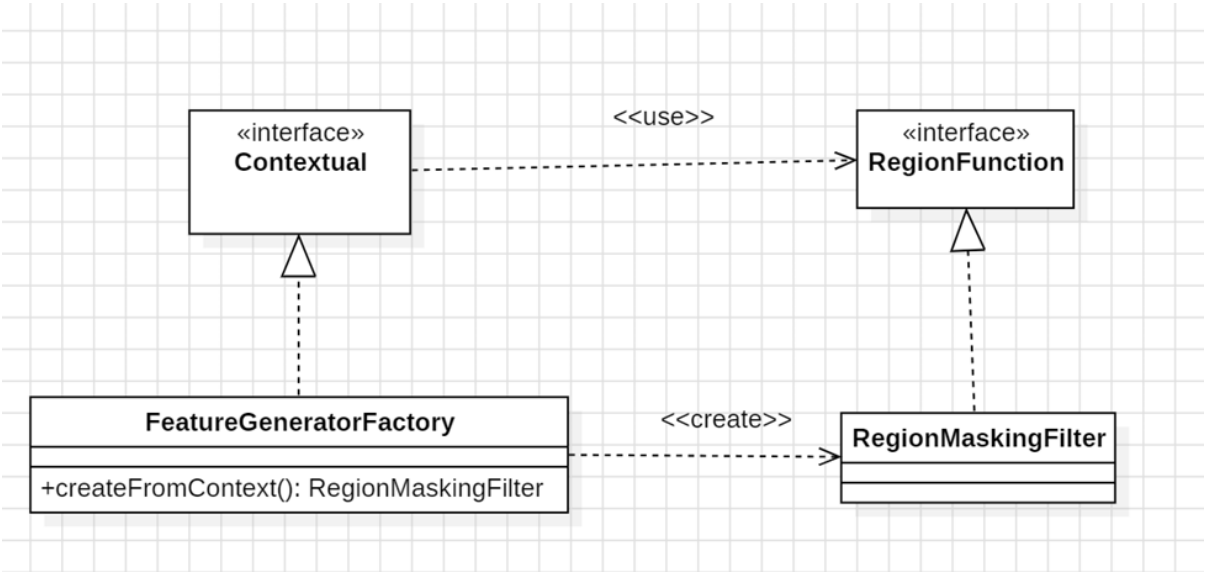
1.1. Design Pattern 1 [Dimitrios Schoinas 65313; review: Diogo Mateus 65379]

Factory method pattern

Code snipet:

```java
public final class FeatureGeneratorFactory implements Contextual<RegionMaskingFilter> {  2 usages  Maddy M
    private final ConfiguredFeatureType type;  3 usages
    private final double density;  2 usages

    public FeatureGeneratorFactory(ConfiguredFeatureType type, double density) {  1 usage  Maddy Miller
        this.type = type;
        this.density = density;
    }

    @Override  Maddy Miller
    public RegionMaskingFilter createFromContext(EditContext input) {
        return new RegionMaskingFilter(
            new NoiseFilter(new RandomNoise(), this.density),
            new FeatureGenerator((EditSession) input.getDestination(), this.type)
        );
    }

    @Override  Maddy Miller
    public String toString() {
        return "feature of type " + type;
    }
}
```

Class diagram:



Location on code base: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit\command\factory\FeatureGeneratorFactory.java createFromContext() method

Discussion of the rationale for identifying: The FeatureGeneratorFactory class uses the createFromContext() method to create a new RegionMaskingFilter instance. This method accepts an EditContext as a parameter, which gives you access to the context needed to construct a RegionMaskingFilter object configured with a noise filter and a feature generator . Instead of directly creating these objects in various parts of the code, the FeatureGeneratorFactory factory encapsulates the creation and allows client code to request configured instances without worrying about the internal details.

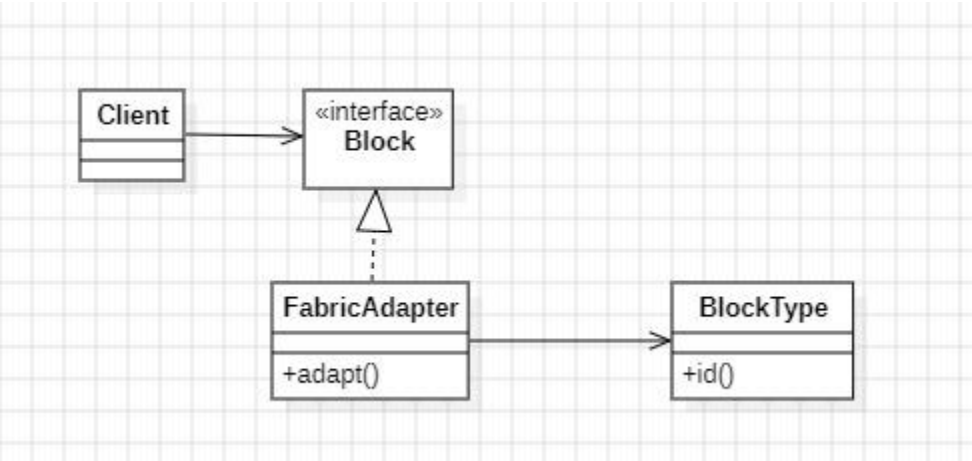
1.2. Design Pattern 2 [[Dimitrios Schoinas 65313; review: João Rivera 62877]

Adapter pattern

Code snipet:

```
220
221 @    public static Block adapt(BlockType blockType) {  Matthew Miller +1
222          return FabricWorldEdit.getRegistry(Registries.BLOCK).get(ResourceLocation.parse(blockType.id()));
223      }
224
225      public static BlockType adapt(Block block) {  Matthew Miller +1
226          return BlockTypes.get(FabricWorldEdit.getRegistry(Registries.BLOCK).getKey(block).toString());
227      }
228
229 @    public static Item adapt(ItemType itemType) {  Matthew Miller +1
230          return FabricWorldEdit.getRegistry(Registries.ITEM).get(ResourceLocation.parse(itemType.id()));
231      }
232
233      public static ItemType adapt(Item item) {  Matthew Miller +1
234          return ItemTypes.get(FabricWorldEdit.getRegistry(Registries.ITEM).getKey(item).toString());
235      }
236
```

Class diagram:



Location on code base: WorldEditPrivate\worldedit-fabric\src\main\java\com\sk89q\worldedit\fabric\FabricAdapter.java (for example: public static Block adapt(BlockType blockType))

Discussion of the rationale for identifying: The Adapter pattern was chosen because it provides a structured and efficient way to convert data between two incompatible systems (Minecraft and WorldEdit) without altering either system. This approach promotes code reuse, maintainability, and scalability while keeping the interfaces clean and compatible.

1.3. Design Pattern 3 [[Dimitrios Schoinas 65313; review: Gonçalo Cascais 60046]

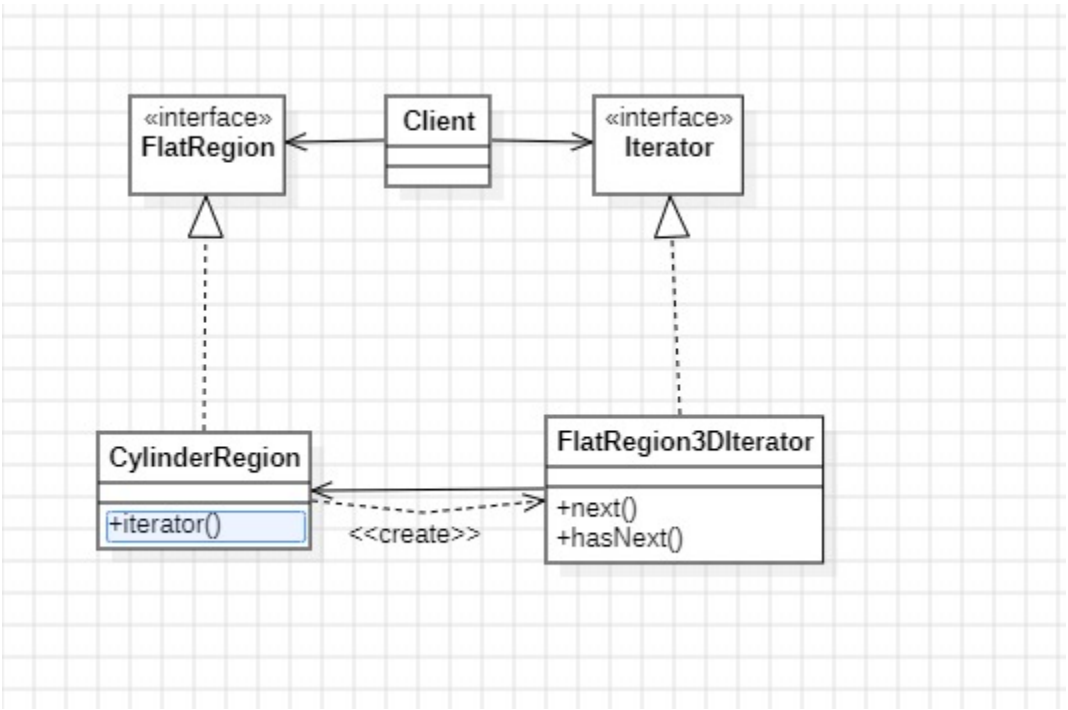
Iterator Pattern

Code snipet:

```java
@Override  Kenzie Togami +1
public Iterator<BlockVector3> iterator() {
    return new FlatRegion3DIterator(this);
}
```

```java
public FlatRegion3DIterator(FlatRegion region) {  2 usages  aumgn
    this(region, region.asFlatRegion().iterator());
}

@Override  aumgn
public boolean hasNext() { return next2D != null; }

@Override  aumgn +2
public BlockVector3 next() {
    if (!hasNext()) {
        throw new NoSuchElementException();
    }
```

Class diagram:



Location on code base: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit\regions\CylinderRegion.java (iterator() method) and WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit\regions\iterator\FlatRegion3DIterator.java

Discussion of the rationale for identifying: In the context of CylinderRegion, this pattern is used to enable traversal of the blocks within a cylindrical region. By providing an iterator, the class lets other parts of the WorldEdit code to process each block in the region without needing to understand how the region is represented internally or how to calculate the points within the cylinder.

1.4. Design Pattern 4 [identification: João Rivera 62877; review: Dimitrios Schoinas 65313]

Singleton structure

Code snippet:

```java
/**
 * Gets the current instance of this class.
 *
 * <p>An instance will always be available, but no platform may yet be
 * registered with WorldEdit, meaning that a number of operations
 * may fail. However, event handlers can be registered.</p>
 *
 * @return an instance of WorldEdit.
 */
public static WorldEdit getInstance() { return instance; }
```

Class diagram:

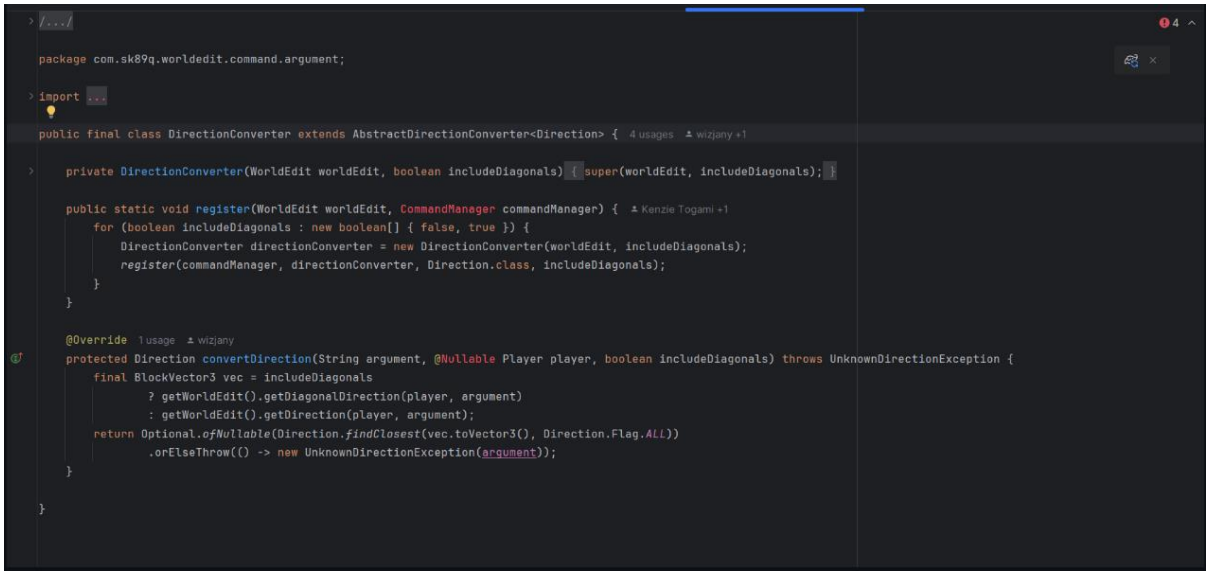
Location on the codebase: worldedit-core/src/main/java/com/sk89q/worldedit/WorldEdit.java method -> getInstance()

Discussion of the rationale for identifying: I initially thought it would make sense to have only a single instance of the mod running so I went to check if that was in fact what happened and it turned out it was. To identify the pattern i searched for a get instance method and made sure the instance itself was made private so that other classes would never get access to it without the getInstance method

1.5. Design Pattern 5 [identification: João Rivera 62877; review: João Nascimento]

Template Method

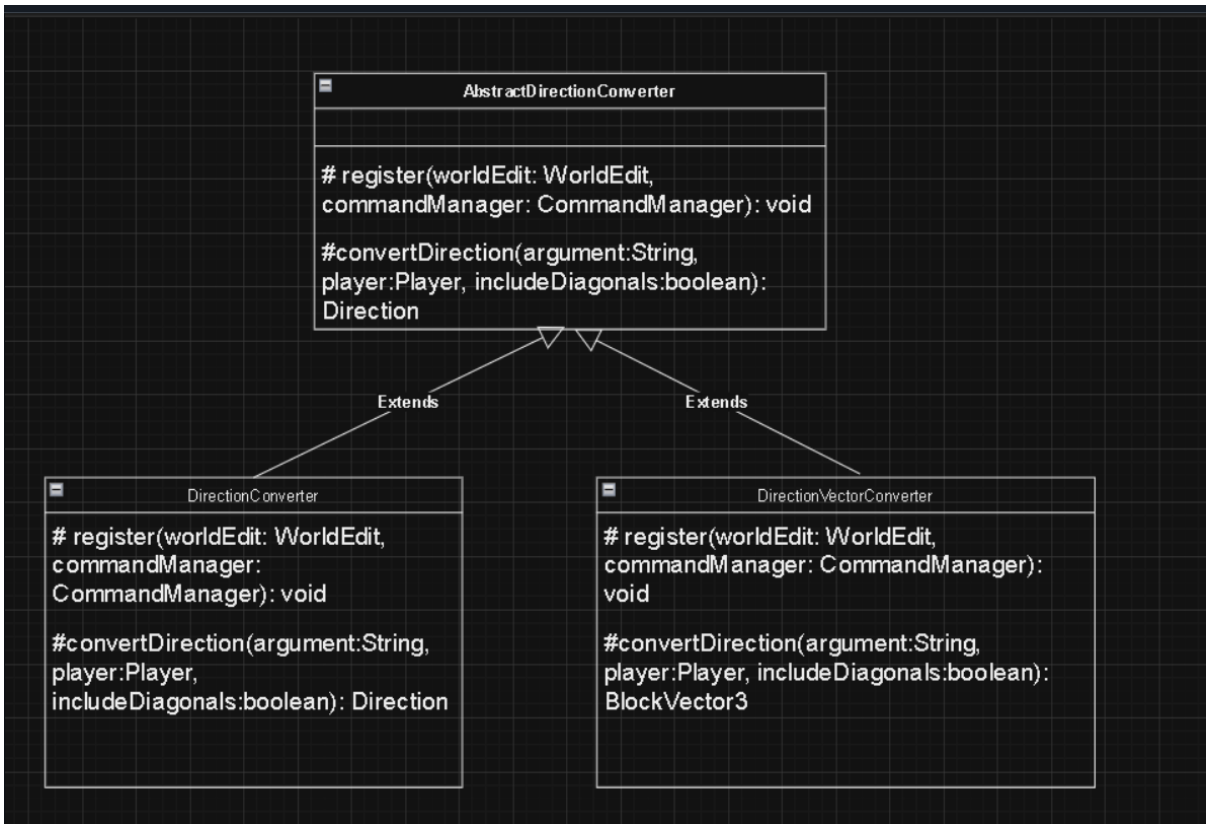
Code snippet:



```
/.../

package com.sk89q.worldedit.command.argument;

import ...

public final class DirectionConverter extends AbstractDirectionConverter<Direction> {  4 usages  wizjany +1

    private DirectionConverter(WorldEdit worldEdit, boolean includeDiagonals) { super(worldEdit, includeDiagonals); }

    public static void register(WorldEdit worldEdit, CommandManager commandManager) {  Kenzie Togami +1
        for (boolean includeDiagonals : new boolean[] { false, true }) {
            DirectionConverter directionConverter = new DirectionConverter(worldEdit, includeDiagonals);
            register(commandManager, directionConverter, Direction.class, includeDiagonals);
        }
    }

    @Override  1 usage  wizjany
    protected Direction convertDirection(String argument, @Nullable Player player, boolean includeDiagonals) throws UnknownDirectionException {
        final BlockVector3 vec = includeDiagonals
                ? getWorldEdit().getDiagonalDirection(player, argument)
                : getWorldEdit().getDirection(player, argument);
        return Optional.ofNullable(Direction.findClosest(vec.toVector3(), Direction.Flag.ALL))
                .orElseThrow(() -> new UnknownDirectionException(argument));
    }

}
```

```
package com.sk89q.worldedit.command.argument;

import ...

public final class DirectionVectorConverter extends AbstractDirectionConverter<BlockVector3> {  6 usages  wizjany +1

    public DirectionVectorConverter(WorldEdit worldEdit, boolean includeDiagonals) {  2 usages  Octavia Togami +1
        super(worldEdit, includeDiagonals);
    }

    public static void register(WorldEdit worldEdit, CommandManager commandManager) {  wizjany
        for (boolean includeDiagonals : new boolean[] { false, true }) {
            DirectionVectorConverter directionConverter = new DirectionVectorConverter(worldEdit, includeDiagonals);
            register(commandManager, directionConverter, BlockVector3.class, includeDiagonals);
        }
    }

    @Override  1 usage  wizjany
    protected BlockVector3 convertDirection(String argument, @Nullable Player player, boolean includeDiagonals) throws UnknownDirectionException {
        return includeDiagonals
                ? getWorldEdit().getDiagonalDirection(player, argument)
                : getWorldEdit().getDirection(player, argument);
    }
}
```

Class diagram:



Location on the codebase: worldedit-core/src/main/java/com/sk89q/worldedit/command/argument/DirectionVectorConverter.java -> DirectionVectorConverter class worldedit-core/src/main/java/com/sk89q/worldedit/command/argument/DirectionConverter.java -> DirectionConverter class worldedit-core/src/main/java/com/sk89q/worldedit/command/argument/AbstractDirectionConverter.java -> AbstractDirectionConverter class

Discussion of the rationale for identifying: The 2 classes that extend the abstract class have the exact same methods just a bit different to fit their needs, very similar functionalities and very similar order of operations, so it seemed to be a case of the Template Method being used.

1.6. Design Pattern 6 [identification: João Rivera 62877; review:Diogo Mateus 65379]

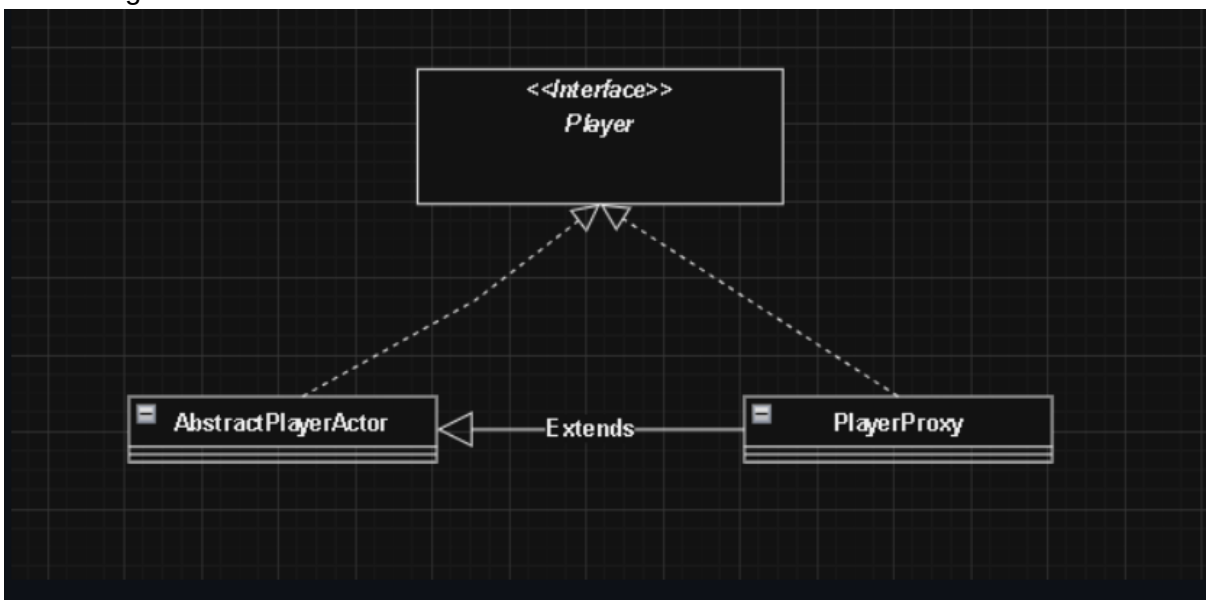
Proxy Pattern

Code snippet:

```java
/.../

package com.sk89q.worldedit.extension.platform;

import ...

class PlayerProxy extends AbstractPlayerActor {  1 usage  sk89q +5

    private final Player basePlayer;  23 usages
    private final Actor permActor;  3 usages
    private final Actor cuiActor;  2 usages
    private final World world;  2 usages

    PlayerProxy(Player basePlayer, Actor permActor, Actor cuiActor, World world) {  1 usage  sk89q +1
        checkNotNull(basePlayer);
        checkNotNull(permActor);
        checkNotNull(cuiActor);
        checkNotNull(world);
        this.basePlayer = basePlayer;
        this.permActor = permActor;
        this.cuiActor = cuiActor;
        this.world = world;
    }

    @Override  8 usages  sk89q
    public UUID getUniqueId() { return basePlayer.getUniqueId(); }

    @Override  30 usages  Matthew Miller
    public BaseItemStack getItemInHand(HandSide handSide) { return basePlayer.getItemInHand(handSide); }

    @Override  2 usages  Matthew Miller
```

```java
@Override  2 usages  sk89q
public BlockBag getInventoryBlockBag() { return basePlayer.getInventoryBlockBag(); }

@Override  sk89q
public String getName() { return basePlayer.getName(); }

@Override  Matthew Miller
public String getDisplayName() { return basePlayer.getDisplayName(); }

@Override  sk89q
public BaseEntity getState() { throw new UnsupportedOperationException("Can't getState() on a player"); }

@Override  sk89q
public Location getLocation() { return basePlayer.getLocation(); }

@Override  3 usages  Matthew Miller
public boolean setLocation(Location location) { return basePlayer.setLocation(location); }

@Override  4 usages  Octavia Togami
public boolean trySetPosition(Vector3 pos, float pitch, float yaw) {
    return basePlayer.trySetPosition(pos, pitch, yaw);
}

@Override  sk89q
public World getWorld() { return world; }

@Override  Matthew Miller +1
@Deprecated
```

Class diagram:



Location on the codebase: worldedit-core/src/main/java/com/sk89q/worldedit/extension/platform/PlayerProxy.java -> PlayerProxy class worldedit-core/src/main/java/com/sk89q/worldedit/extension/platform/AbstractPlayerActor.java -> AbstractPlayerActor class worldedit-core/src/main/java/com/sk89q/worldedit/entity/Player.java -> Player interface

Discussion of the rationale for identifying: Since the class name was what it was i checked it and realized every method in the player proxy was using another method also in the abstractplayeractor, so it seemed to me to be a proxy pattern

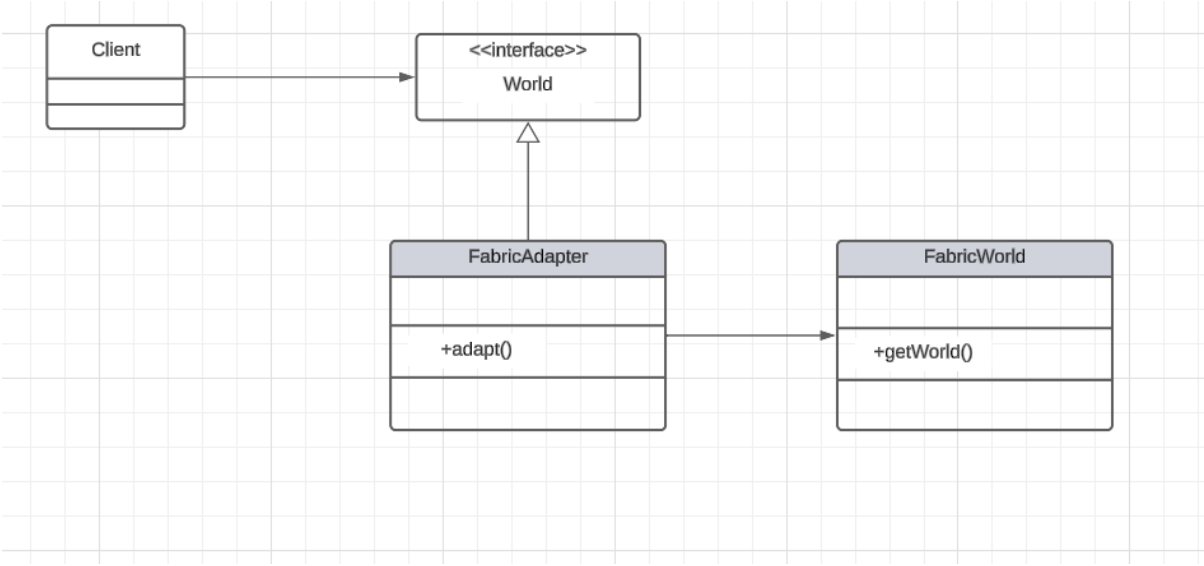
1.7. Design Pattern 7 [identification: Gonçalo Cascais 60046; review: João Rivera 62877]

Adapter pattern

Code snippet:

```java
public static World adapt(net.minecraft.world.level.Level world) {
    return new FabricWorld(world);
}

/**
 * Create a Fabric world from a WorldEdit world.
 *
 * @param world the WorldEdit world
 * @return a Fabric world
 */
public static net.minecraft.world.level.Level adapt(World world) {
    checkNotNull(world);
    if (world instanceof FabricWorld) {
        return ((FabricWorld) world).getWorld();
    } else {
        // TODO introduce a better cross-platform world API to match more easily
        throw new UnsupportedOperationException("Cannot adapt from a " + world.getClass());
    }
}
```

Class Diagram:



Exact location: WorldEditPrivate/worldedit-fabric/src/main/java/com/sk89q/worldedit/fabric/FabricAdapter.java

Discussion of the rationale for identifying: The FabricAdapter, as a Adapter pattern, helps the communication between two incompatible systems. In this case the systems are the minecraft and the WorldEdit.

It receives a World object (from the WorldEdit API) and checks if it is an instance of FabricWorld. If it is, this means that the World is already adapted for WorldEdit, and the method retrieves the original Level contained within FabricWorld.

1.8. Design Pattern 8 [identification: Gonçalo Cascais 60046; review: João Nascimento 62896]

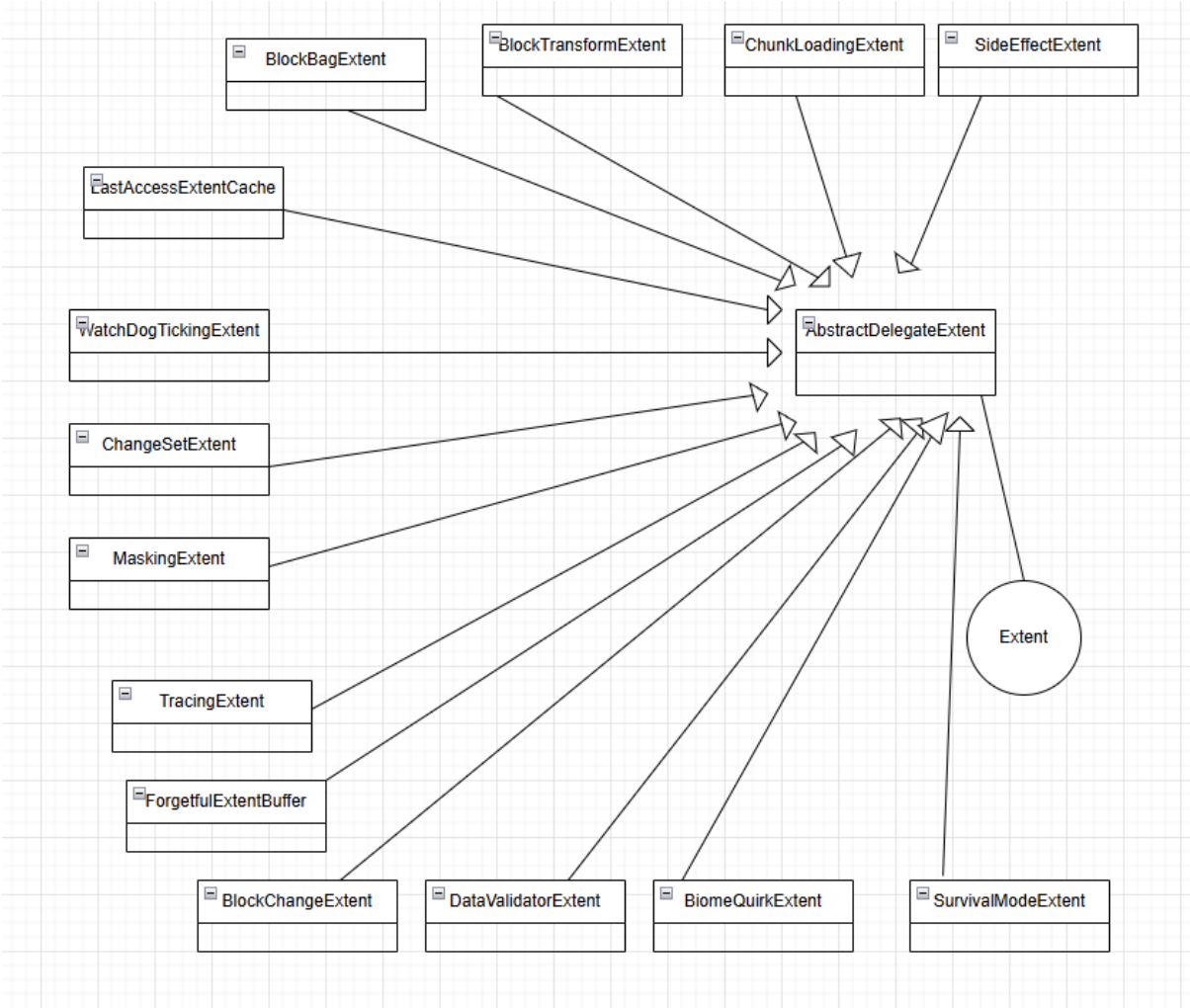
Template Pattern

Code snippet:

```java
public abstract class AbstractDelegateExtent implements Extent {  20 inheritors  sk89q +3

    private final Extent extent;  14 usages

    /**
     * Create a new instance.
     *
     * @param extent the extent
     */
    protected AbstractDelegateExtent(Extent extent) {  sk89q
        checkNotNull(extent);
        this.extent = extent;
    }

    /**
     * Get the extent.
     *
     * @return the extent
     */
    public Extent getExtent() { return extent; }

    @Override  4 overrides  Kenzie Togami +1
    public BlockState getBlock(BlockVector3 position) { return extent.getBlock(position); }

    @Override  4 overrides  Kenzie Togami +1
    public BaseBlock getFullBlock(BlockVector3 position) { return extent.getFullBlock(position); }

    @Override  19 overrides  Kenzie Togami +1
    public <T extends BlockStateHolder<T>> boolean setBlock(BlockVector3 location, T block) throws WorldEditException {
        return extent.setBlock(location, block);
    }

    @Override  3 overrides  sk89q
    @Nullable
    public Entity createEntity(Location location, BaseEntity entity) { return extent.createEntity(location, entity); }

    @Override  2 overrides  sk89q
    public List<? extends Entity> getEntities() { return extent.getEntities(); }
```

```java
@Override  2 overrides  sk89q
public List<? extends Entity> getEntities(Region region) { return extent.getEntities(region); }

@Override  4 usages  Octavia Togami
public boolean fullySupports3DBiomes() { return extent.fullySupports3DBiomes(); }

@Override  1 override  Matthew Miller +1
public BiomeType getBiome(BlockVector3 position) { return extent.getBiome(position); }

@Override  8 overrides  Matthew Miller +1
public boolean setBiome(BlockVector3 position, BiomeType biome) { return extent.setBiome(position, biome); }

@Override  Kenzie Togami +1
public BlockVector3 getMinimumPoint() { return extent.getMinimumPoint(); }

@Override  Kenzie Togami +1
public BlockVector3 getMaximumPoint() { return extent.getMaximumPoint(); }

protected Operation commitBefore() { return null; }

@Override  4 usages  sk89q
public final @Nullable Operation commit() {
    Operation ours = commitBefore();
    Operation other = extent.commit();
    if (ours != null && other != null) {
        return new OperationQueue(ours, other);
    } else if (ours != null) {
        return ours;
    } else if (other != null) {
        return other;
    } else {
        return null;
    }
}
```

Class diagram:



Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/extent/AbstractDelegateExtent

Discussion of the rationale for identifying: Using the Template Method in AbstractDelegateExtent is crucial to ensure consistency, common code reusability, and extensibility when dealing with different types of properties. This results in a more robust, scalable, and maintainable design, allowing new types of properties to be easily added to the system without breaking or duplicating existing logic.

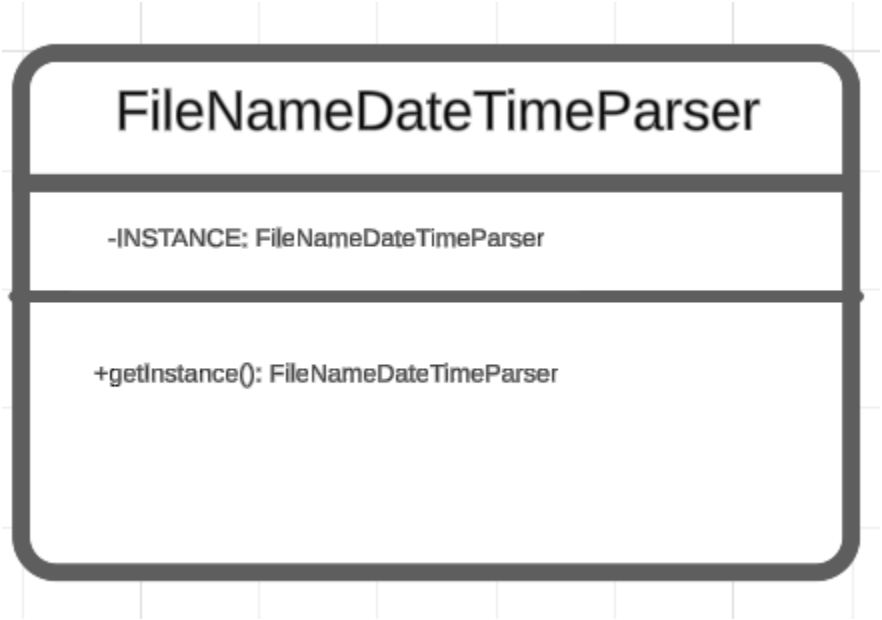
1.9. Design Pattern 9 [identification: Gonçalo Cascais 60046; review: Dimitrios Schoinas]

Singleton pattern

Code snippet:

```
private static final FileNameDateTimeParser INSTANCE = new FileNameDateTimeParser();

public static FileNameDateTimeParser getInstance() {
    return INSTANCE;
}
```

Class diagram:



Exact location: WorldEditPrivate/worldedit-core/src/main/java/com/sk89q/worldedit/util/time/FileNameDateTimeParser.java

Discussion of the rationale for identifying: I was looking through the code when I saw a reference to a calendar and thought it would make sense to just have an instance of a calendar. I checked and found the FileNameDateTimeParser class.

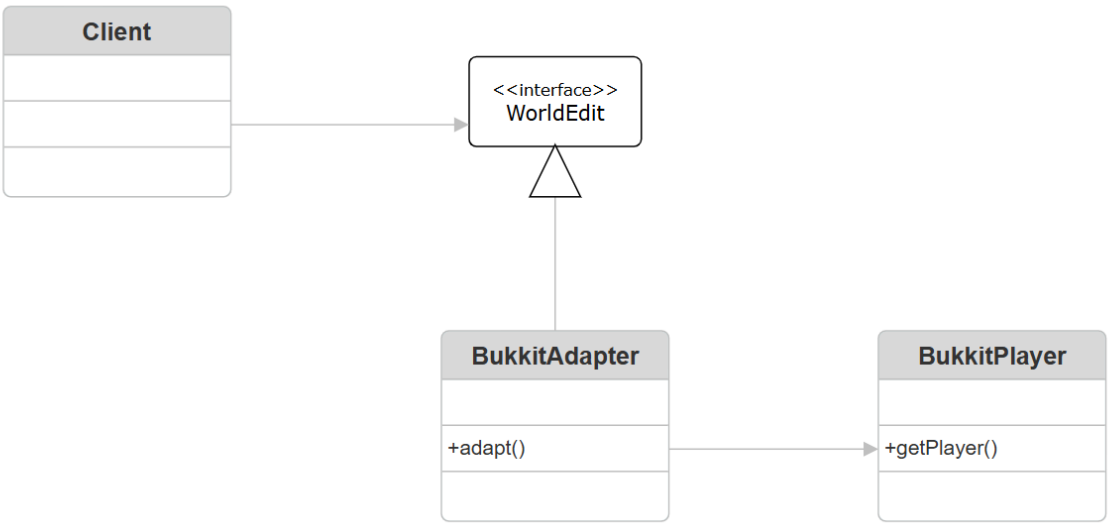
1.10. Design Pattern 10 [identification: Diogo Mateus 65379; review: Dimitrios Schoinas 65313]

Adapter pattern

Code snippet:

```java
     * @return The Bukkit player
     */
    public static Player adapt(com.sk89q.worldedit.entity.Player player) { return ((BukkitPlayer) player).getPlayer(); }

    /**
     * Create a WorldEdit Direction from a Bukkit BlockFace.
     *
     * @param face the Bukkit BlockFace
     * @return a WorldEdit direction
     */
    public static Direction adapt(@Nullable BlockFace face) {  Matthew Miller
        if (face == null) {
            return null;
        }
        switch (face) {
            case NORTH: return Direction.NORTH;
            case SOUTH: return Direction.SOUTH;
            case WEST: return Direction.WEST;
            case EAST: return Direction.EAST;
            case DOWN: return Direction.DOWN;
            case UP:
            default:
                return Direction.UP;
        }
    }

    /**
     * Create a Bukkit world from a WorldEdit world.
     *
     * @param world the WorldEdit world
     * @return a Bukkit world
     */
    public static org.bukkit.World adapt(World world) {  sk89q +1
        checkNotNull(world);
        if (world instanceof BukkitWorld) {
            return ((BukkitWorld) world).getWorld();
```

Class Diagram:



Exact location: worldedit-bukkit/src/main/java/com/sk89q/worldedit/bukkit/BukkitAdapter.java

Discussion of the rationale for identifying: The BukkitAdapter, as a Adapter pattern, helps the communication between two

incompatible systems. In this case the systems are the Bukkit system and the WorldEdit. The Strategy acts as a bridge, making it easier to integrate new or existing code without modifying original implementations.

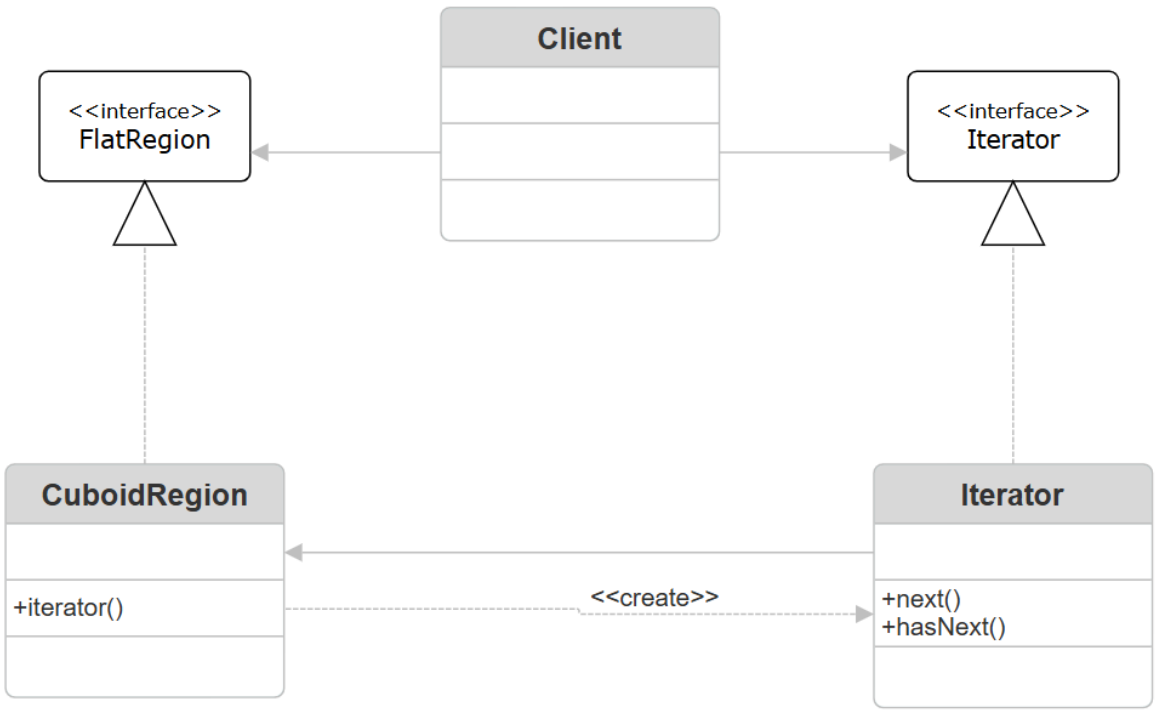
1.11. Design Pattern 11 [identification: Diogo Mateus 65379; review: João Rivera 62877]

Iterator pattern

Code snippet:

```
367     @Override
368     public Iterable<BlockVector2> asFlatRegion() {
369         return () -> new Iterator<BlockVector2>() {
370             private final BlockVector3 min = getMinimumPoint();
371             private final BlockVector3 max = getMaximumPoint();
372             private int nextX = min.x();
373             private int nextZ = min.z();
374 
375             @Override
376             public boolean hasNext() { return (nextX != Integer.MIN_VALUE); }
379 
380             @Override
381             public BlockVector2 next() {
382                 if (!hasNext()) {
383                     throw new NoSuchElementException();
384                 }
385                 BlockVector2 answer = BlockVector2.at(nextX, nextZ);
386                 if (++nextX > max.x()) {
387                     nextX = min.x();
388                     if (++nextZ > max.z()) {
389                         nextX = Integer.MIN_VALUE;
390                     }
391                 }
392                 return answer;
393             }
394         };
395     }
```

Class Diagram:



Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/regions/Polygonal2DRegion.java

Discussion of the rationale for identifying: This code defines an strategy for iterating over a flat, two-dimensional grid within a three-dimensional space. It gives access of each BlockVector to other Classes. When done like this, the iterator hides the internal structure of the data, meaning that you can run through the BlockVectors without needing to know the implementation details of the Polygonal2DRegion Class.

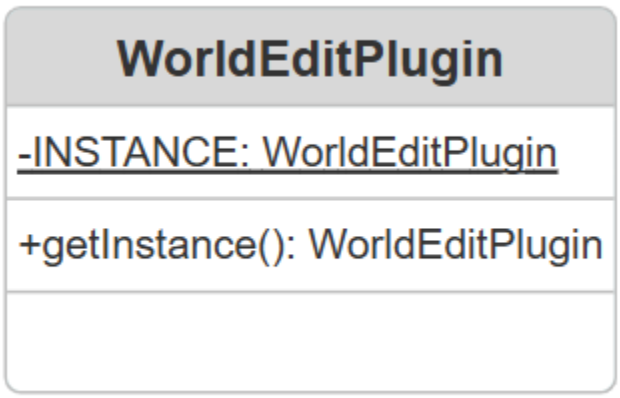
1.12. Design Pattern 12 [identification: Diogo Mateus 65379; review: Gonçalo Cascais 60046]
Singleton pattern

Code snippet:

```
    private static final Logger LOGGER = LogManage
    public static final String CUI_PLUGIN_CHANNEL
    private static WorldEditPlugin INSTANCE;  2 usag
    private static final int BSTATS_PLUGIN_ID = 33

    private final SimpleLifecycled<BukkitImplAdapt
```

```
502     /**
503      * Gets the instance of this plugin.
504      *
505      * @return an instance of the plugin
506      * @throws NullPointerException if the plugin hasn't been enabled
507      */
508     static WorldEditPlugin getInstance() { return checkNotNull(INSTANCE); }
511
```

Class Diagram:



Exact location: worldedit-bukkit/src/main/java/com/sk89q/worldedit/bukkit/WorldEditListener.java

Discussion of the rationale for identifying: As it is a Singleton pattern, this code makes sure only one instance of WorldEditPlugin is created. This brings a variety of good things to the code. Most important of all, it creates a simple and consistent way to access this part of the overall system.

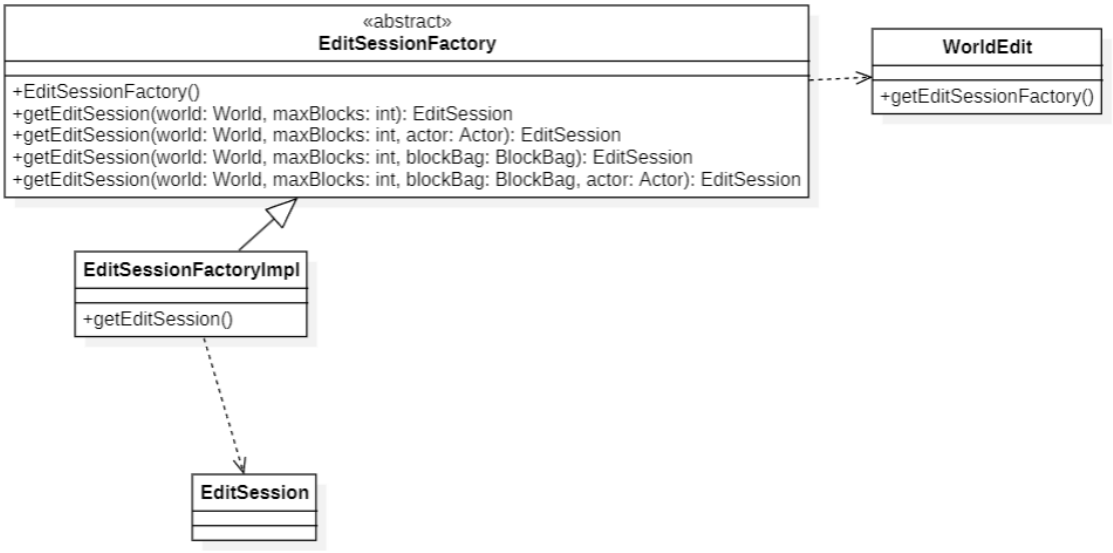
1.13. Design Pattern 13 [identification: João Nascimento 62896; review: João Rivera 62877]

Factory Pattern

Code snippet:

```java
    /**
     * Internal factory for {@link EditSession}s.
     */
    static final class EditSessionFactoryImpl extends EditSessionFactory {

        /**
         * Create a new factory.
         */
        EditSessionFactoryImpl() {
        }

        @Override
        public EditSession getEditSession(World world, int maxBlocks) {
            return getEditSession(world, maxBlocks, null, null);
        }

        @Override
        public EditSession getEditSession(World world, int maxBlocks, Actor actor) {
            return getEditSession(world, maxBlocks, null, actor);
        }

        @Override
        public EditSession getEditSession(World world, int maxBlocks, BlockBag blockBag) {
            return getEditSession(world, maxBlocks, blockBag, null);
        }

        @Override
        public EditSession getEditSession(World world, int maxBlocks, BlockBag blockBag, Actor actor) {
            return WorldEdit.getInstance().newEditSessionBuilder()
                .world(world)
                .maxBlocks(maxBlocks)
                .blockBag(blockBag)
                .actor(actor)
                .build();
        }
```

Class diagram:



Location on code base: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit

Discussion of the rationale for identifying: The importance of this Factory Pattern lies in providing a more robust, flexible and easy-to-maintain design. Tt facilitates consistent EditSession creation by centralizing the logic, making the system more adaptable to change, and improving interoperability between different parts of the code.

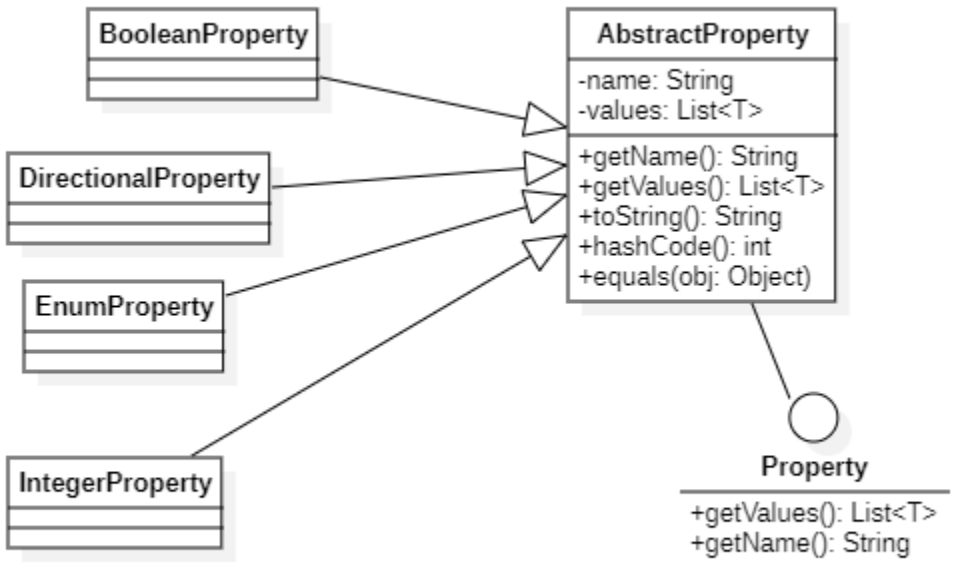
1.14. Design Pattern 14 [identification: João Nascimento 62896; review: Gonçalo Cascais 60046]

Template Method Pattern

Code snippet:

```java
public class MethodHandleEventHandler extends EventHandler {

    private final MethodHandle methodHandle;
    private final String methodName;
    private final Object object;

    /**
     * Create a new event handler that uses MethodHandles to dispatch.
     *
     * @param priority the priority
     * @param object The object to invoke it on
     * @param methodHandle The handle to invoke
     * @param methodName The name of the method (for equality checks)
     */
    protected MethodHandleEventHandler(Priority priority, Object object, MethodHandle methodHandle, String methodName) {
        super(priority);

        this.object = object;
        this.methodHandle = methodHandle.bindTo(object).asType(MethodType.methodType(void.class, Object.class));
        this.methodName = methodName;
    }

    @Override
    public void dispatch(Object event) throws Exception {
        try {
            this.methodHandle.invokeExact(event);
        } catch (Exception | Error e) {
            throw e;
        } catch (Throwable t) {
            // If it's not an Exception or Error, throw it wrapped.
            throw new Exception(t);
        }
    }

    @Override
    public int hashCode() {
        return Objects.hash(methodName, object);
    }
```

Class diagram:



Location on code base: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit\registry\state
Discussion of the rationale for identifying: Using the Template Method in AbstractProperty is crucial to ensure consistency, common code reusability, and extensibility when dealing with

different types of properties. This results in a more robust, scalable, and maintainable design, allowing new types of properties to be easily added to the system without breaking or duplicating existing logic.

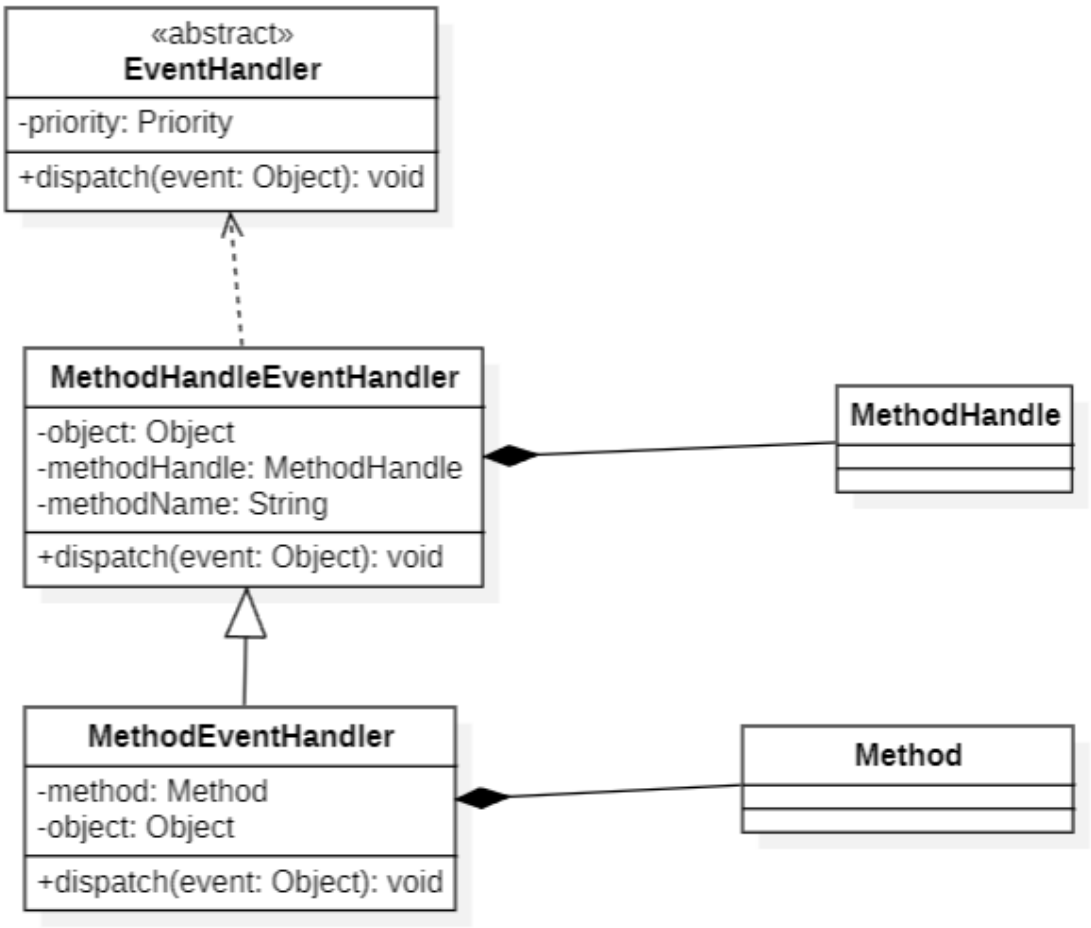
1.15. Design Pattern 15 [identification: João Nascimento 62896; review: Diogo Mateus 65379]

Observer Pattern

Code snippet:

```java
public class MethodHandleEventHandler extends EventHandler {

    private final MethodHandle methodHandle;
    private final String methodName;
    private final Object object;

    /**
     * Create a new event handler that uses MethodHandles to dispatch.
     *
     * @param priority the priority
     * @param object The object to invoke it on
     * @param methodHandle The handle to invoke
     * @param methodName The name of the method (for equality checks)
     */
    protected MethodHandleEventHandler(Priority priority, Object object, MethodHandle methodHandle, String methodName) {
        super(priority);

        this.object = object;
        this.methodHandle = methodHandle.bindTo(object).asType(MethodType.methodType(void.class, Object.class));
        this.methodName = methodName;
    }

    @Override
    public void dispatch(Object event) throws Exception {
        try {
            this.methodHandle.invokeExact(event);
        } catch (Exception | Error e) {
            throw e;
        } catch (Throwable t) {
            // If it's not an Exception or Error, throw it wrapped.
            throw new Exception(t);
        }
    }

    @Override
    public int hashCode() {
        return Objects.hash(methodName, object);
    }
```

Class diagram:



Location on code base: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\util\eventbus

Discussion of the rationale for identifying: Implementing the Observer Pattern with MethodHandleEventHandler is crucial for a robust, decoupled, and efficient event system. It provides greater flexibility, scalability, and reliability, ensuring that different parts of the system can interact in a fluid and modular manner, even in large-scale projects.

2. Codebase Metrics Assessment

File with metrics of the whole group: metricasTodas.xlsx

Metrics from: Dimitios Schoinas 65313 (Reviewed by Gonçalo Cascais 60046)

Metric set: Chidamber-Kemerer Metrics Set

Metrics file: worldedit2.xlsx

Report: Weighted Methods per Class (WMC): Measures the sum of complexities for all methods in a class. Higher WMC values indicate more complex classes with numerous or complex methods. High WMC suggests that a class is doing too much, which may lead to difficulties in understanding and maintaining the class. Classes with lower WMC are generally simpler and more focused. High WMC values can suggest potential code smells such as Divergent Class (a class that has too many responsibilities) and Long Method. Reducing WMC through method extraction or refactoring could enhance readability and maintainability.

Depth of Inheritance Tree (DIT): Represents the maximum inheritance depth of a class within the hierarchy. Higher values mean a class is further down in the inheritance hierarchy. Higher DIT values indicate more inheritance and potential for reuse, but may also introduce greater complexity. A deep inheritance tree (high DIT) can make code more challenging to understand and maintain. A potential code smell is for example Refused Bequest (when subclasses do not use inherited methods). Monitoring DIT helps to ensure inheritance is used appropriately, enhancing reusability without adding unnecessary complexity.

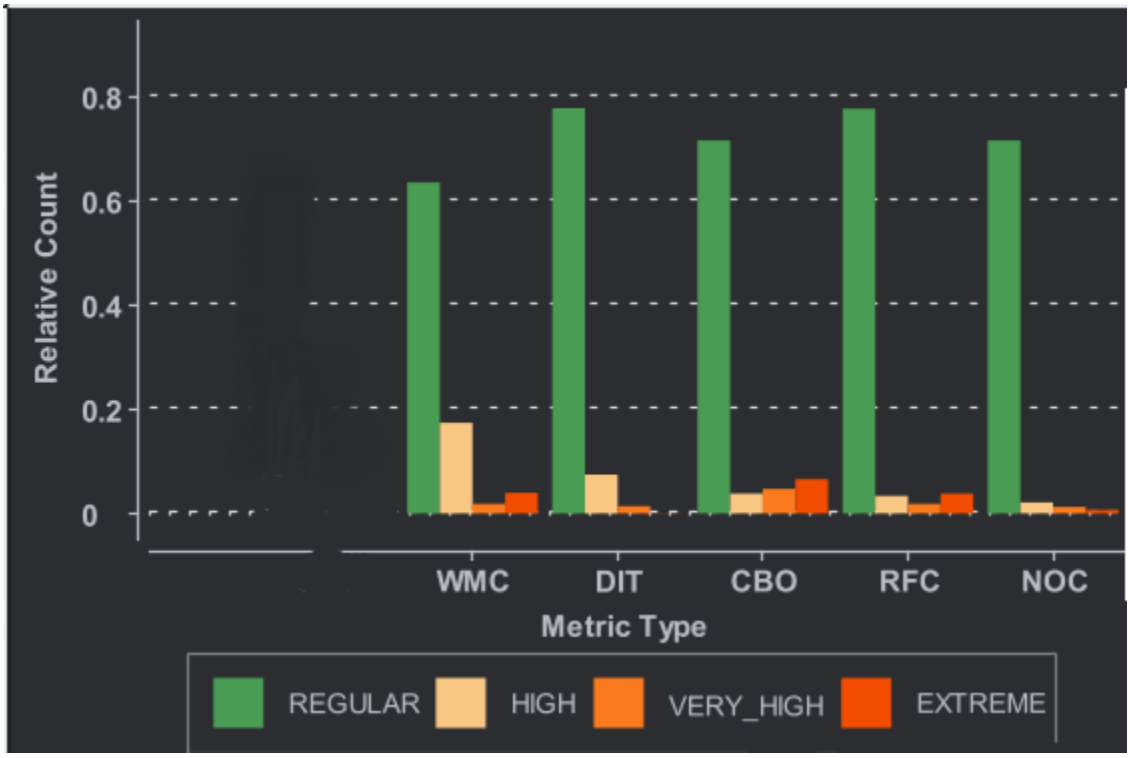
Number of Children (NOC): Counts the direct subclasses a class has. High NOC indicates that a class is a parent to many subclasses, which can suggest it is widely reused or generalized. A high NOC can sometimes point to Speculative Generality (overly generalized classes created with future use cases in mind) and potential Divergent class aspects (excessive responsibility in a single class). Classes with fewer children are often more focused, while classes with numerous children should be reviewed for appropriate use of inheritance versus composition.

Coupling Between Object Classes (CBO): Measures the number of classes that a class is coupled with. High CBO indicates strong dependencies between classes. Higher coupling makes code harder to change because modifications in one class can affect many others. High CBO is often associated with Feature Envy (a class that overly relies on data or functionality in other classes) and Inappropriate Intimacy (classes too familiar with each other's details). Reducing CBO through refactoring or introducing interfaces can improve modularity and make maintenance easier.

Response for a Class (RFC): Counts the methods that can be invoked in response to a message sent to a class, including both its own methods and methods of other classes it calls. High RFC values indicate that a class has a large “surface area” of functionality, making it potentially more complex to understand and test. High RFC values can suggest a Divergent Class (if the class has many methods and interacts heavily with others) or Feature Envy (if it relies on methods from other classes excessively). Monitoring RFC can guide simplifying class interactions to improve readability and testability.

Lack of Cohesion in Methods (LCOM): Measures the cohesiveness of methods within a class. Higher values indicate lower cohesion, meaning methods within a class are less likely to use the same fields or methods. High LCOM suggests that a class may be handling unrelated responsibilities and is a candidate for refactoring. High LCOM values are indicators of Large Class . Improving cohesion by breaking up large classes can help align with the Single Responsibility Principle, simplifying maintenance.

Potential trouble spots in the codebase:



Based on the collected metrics, I identified potential trouble spots in the codebase by examining extreme values through the graphics and metrics values collected. These visualizations allowed me to see the distribution of each metric and pinpoint outliers. For example, classes like EditSession and FabricRegistries had significantly high WMC and RFC values, suggesting they may be Divergent Classes with too many

responsibilities. Classes such as BlockVector3 had high CBO values, indicating a strong dependency on other classes, which could make it difficult to modify without affecting other parts of the system. Classes with high LCOM values, like OffsetsMask2D.Builder, show low cohesion and may be handling multiple responsibilities, suggesting they may benefit from refactoring to improve cohesion and follow the Single Responsibility Principle. Addressing these trouble spots could enhance code readability, maintainability, and reduce potential bugs.

By examining the metrics in conjunction with identified code smells, we can see clear relationships: The high CBO (Coupling Between Object Classes) and low cohesion (High LCOM) values point to potential Inappropriate Intimacy by indicating strong interdependencies and scattered responsibilities. The high WMC(Weighted Methods per Class) and high RFC(Response for a Class) suggest Long Method issues, as they reflect classes with complex methods that could be simplified . Also I want to say that the graph shows that there might be a lot of code smells related to the CBO metric since it's extreme bar is very high compared to the others.

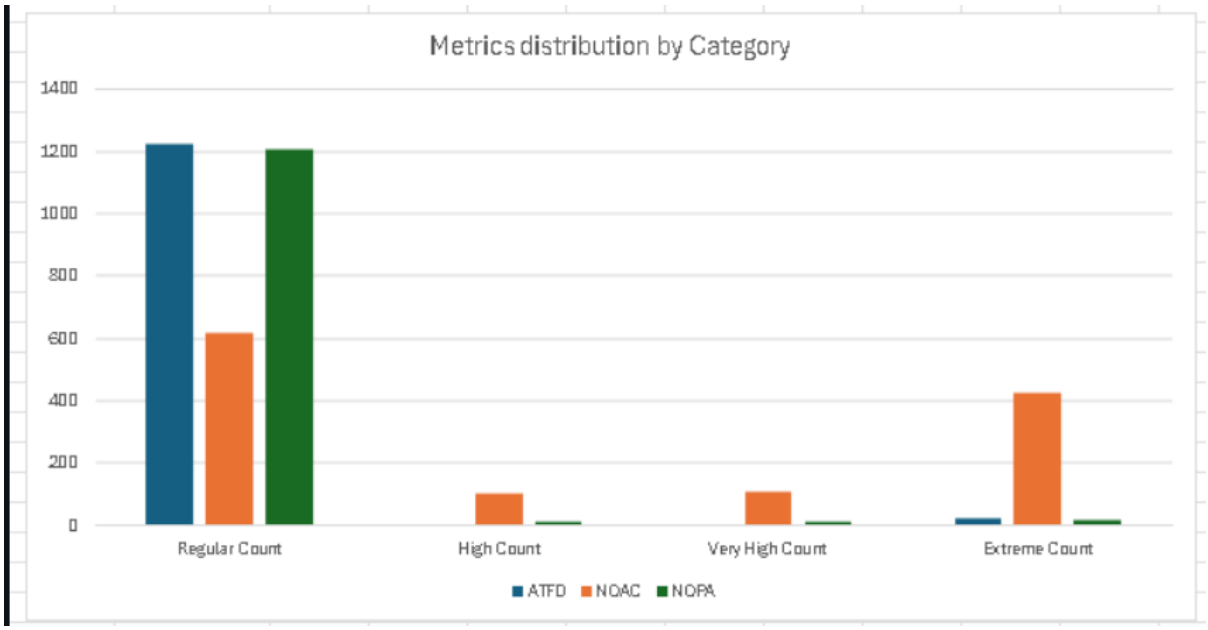
Relation with the identified code smells: I found the Inaproppriate Intimacy code smell because of the collected code metrics, the metric CBO(Coupling Between Objects) was way higher in the class ToolCommands compared to the average number between all classes. I found the Long method code smell with the collected code metrics because the value of the WMC(Weighted Method Count) metric was almost double the avarage class value, so for me it was a clue that it could be a code smell.

Metrics by: João Rivera 62877 (Reviewed by Dimitrios Schoinas 65313)

Metrics set used: Lanza-Marinescu metrics set

Metrics from the metrics set: -Access To Foreign Data (ATFD)
-Number Of Public Attributes(NOPA)
-Number Of Accessor Methods(NOAC)

Graph:



ATFD metric: This metric measures the amount of attributes from other classes directly accessed by the class being measured. A high ATFD value can indicate high coupling between classes, a low ATFD value is desirable as it implies better encapsulation and modularity. 3 of the classes that show an extreme ATFD value is the DefaultBlockParser class(worldedit-core/src/main/java/com/sk89q/worldedit/extension/factory/parser/DefaultBlockParser.java), the UtilityCommands class(worldedit-core/src/main/java/com/sk89q/worldedit/command/UtilityCommands.java) and the BrushCommands class (worldedit-core/src/main/java/com/sk89q/worldedit/command/BrushCommands.java). A high value of this metric is also usually associated to the feature envy code smell.

NOPA metric: This metric measures the number of attributes within a class that are publicly accessible. Public attributes are those that can be accessed or modified directly by other classes, without the need for any access methods (getters/setters). A high value of the NOPA metric can indicate bad encapsulation and bring unintended side effects or coupling between classes, which makes the system harder to extend. Also good design principles suggest using private attributes and using setter/getter methods to access these attributes. The BlockMap class (worldedit-core/src/main/java/com/sk89q/worldedit/util/collection/BlockMap.java) shows an extreme level of NOPA. A high value of this metric is also usually associated to the data clump and god class code smells.

NOAC metric: This metric refers to the number of getter and setter methods (accessors) that a class provides to access or modify its private or protected

attributes. These methods are typically used to access the attributes of a class indirectly, promoting encapsulation and data hiding. A high NOAC value could signal that said class is doing too much or its internal state is poorly encapsulated and a low could suggest insufficient access to class attributes or incomplete encapsulation. This WorldEdit project has an extremely high amount of extreme NOAC values, with the biggest outliers being by far the EditSession class (worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java). High values of this metric is usually associate with the excessive getters and setters and god object code smells.

Metrics by: Gonçalo Cascais 60046 (Reviewed by João Rivera 62877)

Metrics set used: Li-Henry metrics set

Metrics from the metrics set: -Number of attributes and methods (size2) -Data abstraction Coupling(DAC) -Number of Methods(NOM) - Message of methods(MPC)

DAC Metric: A high DAC (Data Abstraction Coupling) value indicates that the class has many dependencies on other classes, meaning it uses many data types that belong to other classes. This implies that the class is heavily coupled with other parts of the system. Such high coupling can have negative implications for the maintainability, flexibility, and reusability of the class. A class with extreme values of DAC is LocalSession(worldedit-core/src/main/java/com.sk89q/worldedit/LocalSession)

MPC Metric: A high MPC (Message Passing Coupling) value indicates that a class makes numerous method calls to other classes. In other words, it frequently relies on methods from other objects to accomplish its tasks, leading to a high level of interaction or coupling between the class and other classes. We have extreme values of MPC in class ItemTypes(worldedit-core/src/main/java/com.sk89q/worldedit/world/item/ItemTypes

SIZE2 Metric: A high SIZE2 metric value indicates a large number of methods within a class. In object-oriented design metrics, SIZE2 specifically measures the total count of methods defined in a class, both inherited and newly created. When SIZE2 is high, it points to a class with many responsibilities or behaviors, which can have several implications. Extreme values of this metric in PaperweightFakePlayer(worldedit-bukkit/adapters(adapter-1.21/sec/main/java/com.sk89q.worldedit.bukkit.adapter.impl.v1_21/PaperweightFakePlayer)

NOM Metric: A high NOM (Number of Methods) value indicates that a class has a large number of methods. This metric is often used to assess the complexity and responsibility of a class in object-oriented programming. When the NOM is high, it suggests that the class is likely managing multiple tasks or contains significant functionality. Extreme values of NOM in class NavigableEditorPane(com.sk89q.worldedit.internal.util.InfoEntryPoint.NavigableEditorPane)

Metrics by: Diogo Mateus 65379 (Reviewed by João Nascimento 62896)

Metric set: Lorenz Kidd Metrics Set

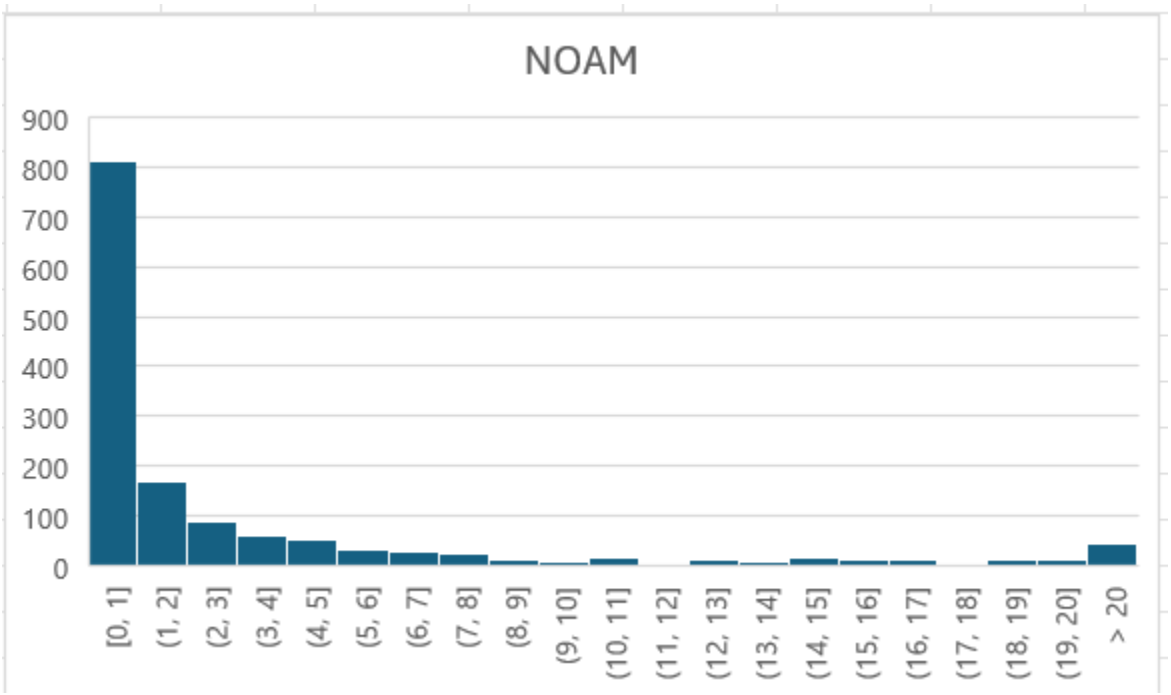
Metrics file: (found in git)

What is represents each Metric:

NOAM -> Number Of Added Methods 1.2) Added Methods: Nem methods that a subclass introduces, which are not inherited or overridden from a superclass.
NOA -> Number Of Attributes
NOOM -> Number Of Overridden Methods 3.1) Overridden Method: A Method in a subclass that has the same name, return type, and parameters as a method in it's superclass.

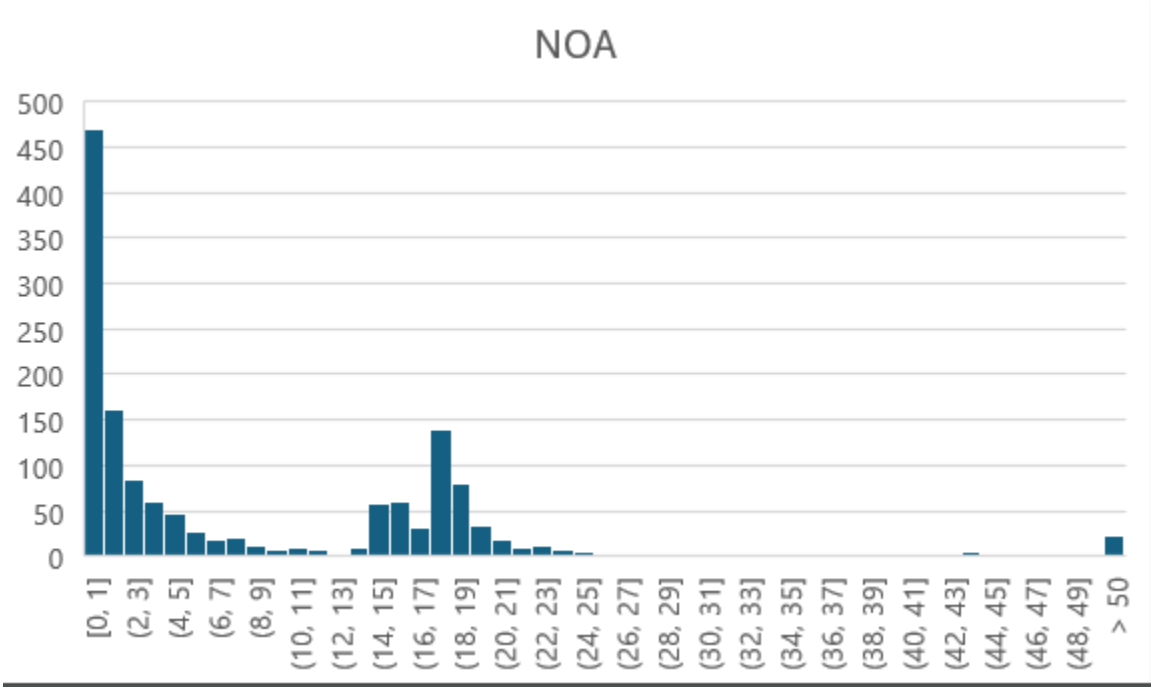
NOO -> Number Of Operations
Analysis of the data:

NOAM:



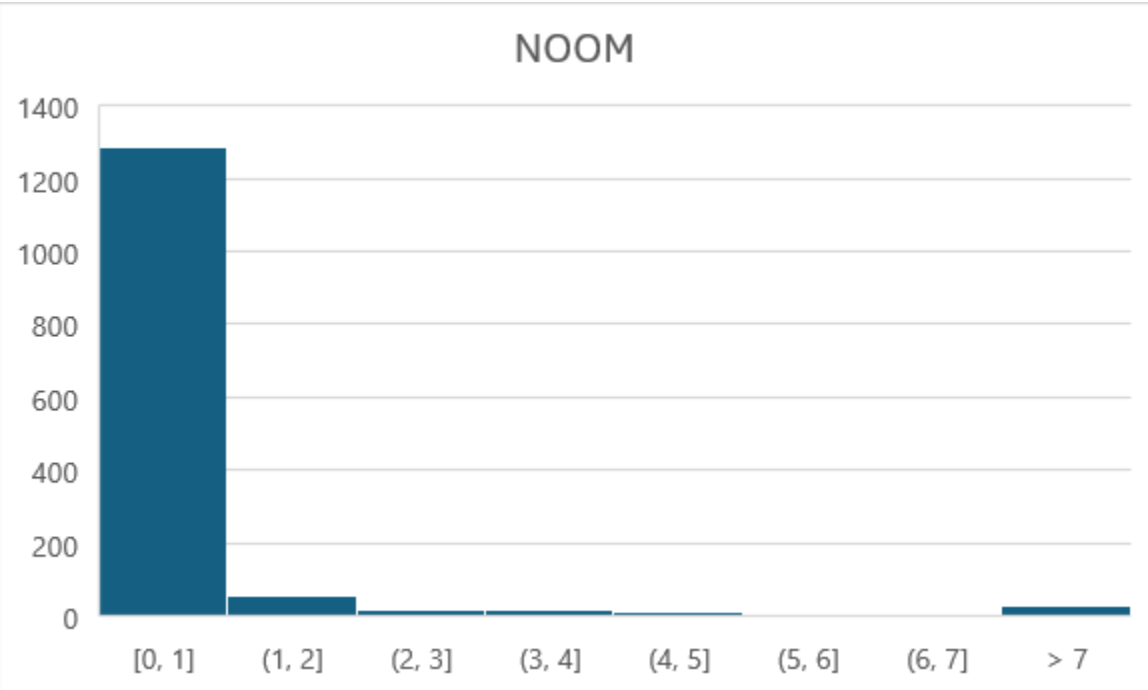
As said, this metric represents the number of Added Methods (1.2). As we can see, is very common for a Method to have zero or one Added Method. This is fine and can be considered good. Most of this methods are specialized for certain purposes and only used in methods inside the Class itself. That said, to many can be a sign for a Code Smell. Can indicate that the methods are divided in to many parts, the class is to complex or that certain methods should be inherited or overridden from a superclass. From the graph we can conclude that arround 70 to 80 classes show at least a sign of this problem.

NOA:



This metric counts the number of attributes of a given Class. We can see that this graph has two tendencies, one to values more low, which is good and expected, and other close to 17, this may be a problem. We want to always have the least amount of attributes as possible, only the ones we need. That said, 17 and 18 is a lot. This may indicate that a large group of classes stores to much data. Maybe it would be better to divide this classes to minimize class responsability and organize the code.

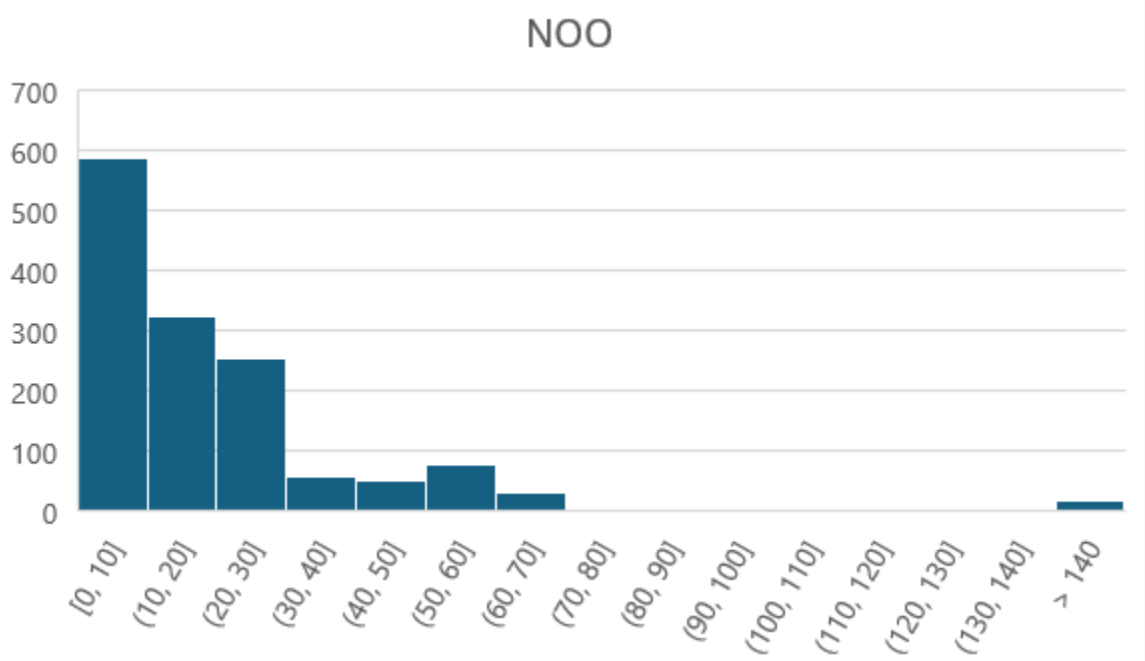
NOOM:



NOOM counts the number of overridden methods (3.1) in a class. This can be beneficial. You can define a generic behavior in a superclass and then override it in subclasses for more specific behaviors. This makes the codebase more reusable by sharing common logic among related classes while customizing where needed. Even than, this can cause

problems if over used. Overriding methods in deep inheritance hierarchies can make code complex and harder to understand, as behaviors might be defined or modified in multiple places. As shown in the graph, the results seem pretty good, with only a few having more than seven.

NOO:



Like shown above, this metric counts the number of operations on a given Class. A high number of operations, like in most cases, can be needed, but we want as little operations as possible. High number of operations indicates a complex and hard code to understand. It can also be a sign that a class takes on to many responsabilities. In this topic, although it could be justified if analysed, I consider that there are arroud 150 in a considerable risk of having Code Smells, with arroud 15 classes in a big risk of having the same.

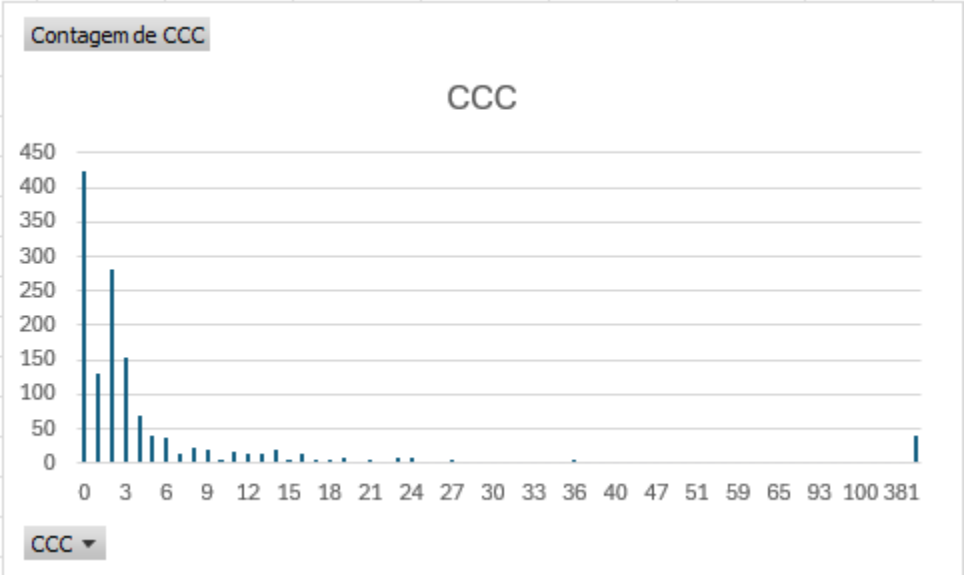
Potential trouble spots: This were the most problematic classes in each metric: NOAM: com.sk89q.worldedit.internal.util.InfoEntryPoint.NavigableEditorPane NOA: com.sk89q.worldedit.world.item.ItemTypes NOOM: com.sk89q.worldedit.internal.expression.invoke.CompilingVisitor NOO: com.sk89q.worldedit.bukkit.adapter.impl.v1_21.PaperweightFakePlayer

---

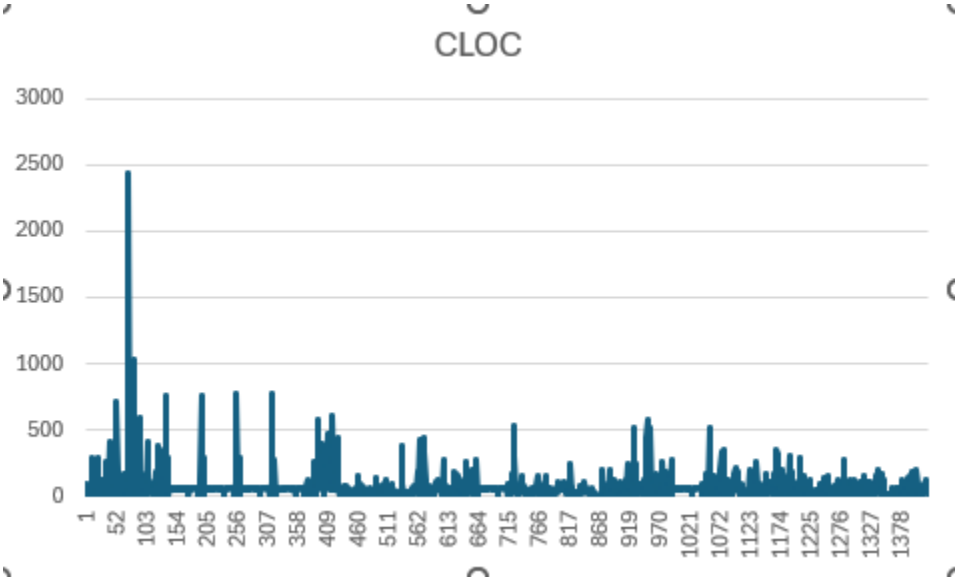
Metrics by: João Nascimento 62896 (reviewed by Diogo Mateus 65379)
Metrics chosen: CCC, CLOC, CMI

(Metrics file found in git)

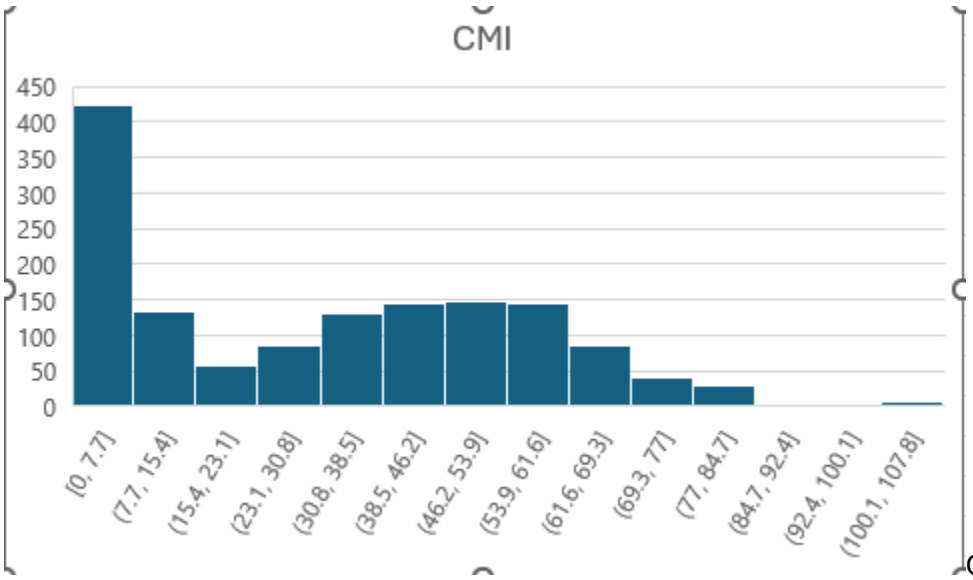
CCC (Classic Cyclomatic Complexity): Measures the cyclomatic complexity of a class (loops and conditions).
High CCC values may lead to a class with very long methods with many conditions and loops, leading to a possible God Class or very long methods code smell.



CLOC (Comment Lines of Code): Measures the comment lines in font code.
High CLOC may lead to a Commented Code, or Excessive Comments, while Low CLOC may lead to a Lack of Documentaion code smell.



CMI (Complexity Measure Index): a combined metric of ciclomatic complexity, cohesion, coupling, etc. giving a global complexity index.
High CMI indicates a highly complex class, meaning high coupling, many methods, low cohesion and others, giving an idea of Spaghetti Code or a God class code smell.



Ok

3. Code Smells

Code smells from: Dimitrios Schoinas 65313

Inaproppriate Intimacy (Reviewed by João Rivera 62877)

Code snipet:

```java
public static void register(CommandRegistrationHandler registration,  Kenzie Togami +2
                            CommandManager commandManager,
                            CommandManagerService commandManagerService,
                            WorldEdit worldEdit) {
    // Collect the tool commands
    CommandManager collect = commandManagerService.newCommandManager();

    registration.register(
        collect,
        ToolCommandsRegistration.builder(),
        new ToolCommands(worldEdit)
    );

    // Register deprecated global commands
    Set<org.enginehub.piston.Command> commands = collect.getAllCommands()
        .collect(Collectors.toSet());
    for (org.enginehub.piston.Command command : commands) {
        if (command.getAliases().contains("unbind")) {
            // Don't register new /tool <whatever> alias
            command = command.toBuilder().aliases(
                Collections2.filter(command.getAliases(), alias -> !"unbind".equals(alias))
            ).build();
        }
        if (command.getName().equals("stacker")) {
            // Don't register /stacker
            continue;
        }
        commandManager.register(CommandUtil.deprecate(
            }
            commandManager.register(CommandUtil.deprecate(
                command,  reason: "Global tool names cause conflicts "
                    + "and will be removed in WorldEdit 8",
                CommandUtil.ReplacementMessageGenerator.forNewCommand(ToolCommands::asNonGlobal)
            ));
        }

        // Remove aliases with / in them, since it doesn't make sense for sub-commands.
        Set<org.enginehub.piston.Command> nonGlobalCommands = commands.stream()
            .map(command ->
                command.toBuilder().aliases(
                    Collections2.filter(command.getAliases(), alias -> !alias.startsWith("/"))
                ).build()
            )
            .collect(Collectors.toSet());
        commandManager.register( name: "tool", command -> {
            command.addPart(SubCommandPart.builder(
                TranslatableComponent.of( key: "tool"),
                TextComponent.of( content: "The tool to bind")
            )
                .withCommands(nonGlobalCommands)
                .required()
                .build());
            command.description(TextComponent.of( content: "Binds a tool to the item in your hand"));

            command.condition(new SubCommandPermissionCondition.Generator(nonGlobalCommands).build());
        });
```

Exact location: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit\command\ToolCommands.java

Explanation of the rationale for identifying this code smell: The ToolCommands class relies on multiple components and accesses many different classes and their specific functions to perform command operations. This is especially evident in methods that use several classes and objects from other packages (CommandManager, LocalSession, WorldEdit, Player, etc.), indicating that ToolCommands has a high level of coupling with these classes. This reliance on the internal workings of many classes suggests that ToolCommands might be too "intimate" with these other classes. I found this code smell because of the collected code metrics, the metric CBO(Coupling Between Objects) was way higher in this class compared to the average number between all classes.

Refactoring: To address the Inappropriate Intimacy smell in this code, we can focus on reducing the direct interactions between ToolUtilCommands and BrushTool by adding a service layer that abstracts these interactions. This can prevent ToolUtilCommands from accessing too many low-level details in BrushTool and LocalSession, reducing coupling and improving modularity.

Long method (Reviewed by Gonçalo Cascais 60046)

Code snipet:

```java
private LinCompoundTag write2(Clipboard clipboard) {  1 usage  Octavia Togami +4
    Region region = clipboard.getRegion();
    BlockVector3 origin = clipboard.getOrigin();
    BlockVector3 min = region.getMinimumPoint();
    BlockVector3 offset = min.subtract(origin);
    int width = region.getWidth();
    int height = region.getHeight();
    int length = region.getLength();

    if (width > MAX_SIZE) {
        throw new IllegalArgumentException("Width of region too large for a .schematic");
    }
    if (height > MAX_SIZE) {
        throw new IllegalArgumentException("Height of region too large for a .schematic");
    }
    if (length > MAX_SIZE) {
        throw new IllegalArgumentException("Length of region too large for a .schematic");
    }

    LinCompoundTag.Builder schematic = LinCompoundTag.builder();
    schematic.putInt( name: "Version", CURRENT_VERSION);
    schematic.putInt(
         name: "DataVersion",
        WorldEdit.getInstance().getPlatformManager().queryCapability(Capability.WORLD_EDITING).getDataVe
    );
```

```java
        metadata.putInt( name: "WEOffsetX", offset.x());
        metadata.putInt( name: "WEOffsetY", offset.y());
        metadata.putInt( name: "WEOffsetZ", offset.z());

        LinCompoundTag.Builder worldEditSection = LinCompoundTag.builder();
        worldEditSection.putString( name: "Version", WorldEdit.getVersion());
        worldEditSection.putString(
             name: "EditingPlatform",
            WorldEdit.getInstance().getPlatformManager().queryCapability(Capability.WORLD_EDITING).id()
        );
        worldEditSection.putIntArray( name: "Offset", new int[] {
            offset.x(), offset.y(), offset.z()
        });

        LinCompoundTag.Builder platformsSection = LinCompoundTag.builder();
        for (Platform platform : WorldEdit.getInstance().getPlatformManager().getPlatforms()) {
            platformsSection.put(
                platform.id(),
                LinCompoundTag.builder()
                    .putString( name: "Name", platform.getPlatformName())
                    .putString( name: "Version", platform.getPlatformVersion())
                    .build()
            );
        }
        worldEditSection.put( name: "Platforms", platformsSection.build());
```

```
        worldEditSection.put( name: "Platforms", platformsSection.build());

        metadata.put( name: "WorldEdit", worldEditSection.build());

        schematic.put( name: "Metadata", metadata.build());

        schematic.putShort( name: "Width", (short) width);
        schematic.putShort( name: "Height", (short) height);
        schematic.putShort( name: "Length", (short) length);

        // The Sponge format Offset refers to the 'min' points location in the world. That's our 'Origin'
        schematic.putIntArray( name: "Offset", new int[] {
            min.x(),
            min.y(),
            min.z(),
        });

        int paletteMax = 0;
        Object2IntMap<String> palette = new Object2IntLinkedOpenHashMap<>();

        LinListTag.Builder<LinCompoundTag> tileEntities = LinListTag.builder(LinTagType.compoundTag());

        ByteArrayOutputStream buffer = new ByteArrayOutputStream( size: width * height * length);

        for (int y = 0; y < height; y++) {
            int y0 = min.y() + y;
```

```
147         for (int y = 0; y < height; y++) {
148             int y0 = min.y() + y;
149             for (int z = 0; z < length; z++) {
150                 int z0 = min.z() + z;
151                 for (int x = 0; x < width; x++) {
152                     int x0 = min.x() + x;
153                     BlockVector3 point = BlockVector3.at(x0, y0, z0);
154                     BaseBlock block = clipboard.getFullBlock(point);
155                     LinCompoundTag nbt = block.getNbt();
156                     if (nbt != null) {
157                         LinCompoundTag.Builder values = nbt.toBuilder();
158
159                         values.remove( name: "id"); // Remove 'id' if it exists. We want 'Id'
160
161                         // Positions are kept in NBT, we don't want that.
162                         values.remove( name: "x");
163                         values.remove( name: "y");
164                         values.remove( name: "z");
165
166                         values.putString( name: "Id", block.getNbtId());
167                         values.putIntArray( name: "Pos", new int[] { x, y, z });
168
169                         tileEntities.add(values.build());
170                     }
171
172                     String blockKey = block.toImmutableState().getAsString();
```

```
                    if (palette.containsKey(blockKey)) {
                        blockId = palette.getInt(blockKey);
                    } else {
                        blockId = paletteMax;
                        palette.put(blockKey, blockId);
                        paletteMax++;
                    }

                    while ((blockId & -128) != 0) {
                        buffer.write( b: blockId & 127 | 128);
                        blockId >>>= 7;
                    }
                    buffer.write(blockId);
                }
            }
        }

        schematic.putInt( name: "PaletteMax", paletteMax);

        LinCompoundTag.Builder paletteTag = LinCompoundTag.builder();
        Object2IntMaps.fastForEach(palette, e -> paletteTag.putInt(e.getKey(), e.getIntValue()));

        schematic.put( name: "Palette", paletteTag.build());
        schematic.putByteArray( name: "BlockData", buffer.toByteArray());
        schematic.put( name: "BlockEntities", tileEntities.build());
```

```
        LinCompoundTag.Builder paletteTag = LinCompoundTag.builder();
        Object2IntMaps.fastForEach(palette, e -> paletteTag.putInt(e.getKey(), e.getIntValue()));

        schematic.put( name: "Palette", paletteTag.build());
        schematic.putByteArray( name: "BlockData", buffer.toByteArray());
        schematic.put( name: "BlockEntities", tileEntities.build());

        // version 2 stuff
        if (clipboard.hasBiomes()) {
            writeBiomes(clipboard, schematic);
        }

        if (!clipboard.getEntities().isEmpty()) {
            LinListTag<LinCompoundTag> value = WriterUtil.encodeEntities(clipboard, positionIsRelative: false);
            if (value != null) {
                schematic.put( name: "Entities", value);
            }
        }

        return schematic.build();
    }

    private void writeBiomes(Clipboard clipboard, LinCompoundTag.Builder schematic) {
```

Exact location: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit\extent\clipboard\io\sponge\SpongeSchematicV2Writer.java

Explanation of the rationale for identifying this code smell: The method is lengthy, spanning numerous lines with multiple embedded blocks and layers of logic. This makes it hard to read, understand, and maintain.It performs several distinct tasks that could be refactored into separate methods. I found this code smell with the collected

code metrics because the value of the WMC(Weighted Method Count) metric was almost double the avarage class value.

Refactoring: To address the Long Method smell in the write2 method, we can refactor it by breaking down sections of logic into smaller, more focused methods, since I can clearly identify segments of the code that can be implemented into their own separate method.

Long parameter list (reviewer: João Nascimento 62896)

Code snipet:

```
public int butcher(Actor actor,
                   @Arg(desc = "Radius to kill mobs in", def = "")
                       Integer radius,
                   @Switch(name = 'p', desc = "Also kill pets")
                       boolean killPets,
                   @Switch(name = 'n', desc = "Also kill NPCs")
                       boolean killNpcs,
                   @Switch(name = 'g', desc = "Also kill golems")
                       boolean killGolems,
                   @Switch(name = 'a', desc = "Also kill animals")
                       boolean killAnimals,
                   @Switch(name = 'b', desc = "Also kill ambient mobs")
                       boolean killAmbient,
                   @Switch(name = 't', desc = "Also kill mobs with name tags")
                       boolean killWithName,
                   @Switch(name = 'f', desc = "Also kill all friendly mobs (Applies the flags `-abgnpt`)
                       boolean killFriendly,
                   @Switch(name = 'r', desc = "Also destroy armor stands")
                       boolean killArmorStands,
                   @Switch(name = 'w', desc = "Also kill water mobs")
                       boolean killWater) throws WorldEditException {
    LocalConfiguration config = we.getConfiguration();

    if (radius == null) {
```

Exact location: WorldEditPrivate\worldedit-core\src\main\java\com\sk89q\worldedit\command\UtilityCommands.java (butcher() method)

Explanation of the rationale for identifying this code smell: This method has too many parameters, making it hard to understand and maintain. I chose this code smell because the metric profile showed that this method had a very high parameter number (note: this metric is not present in my metrics file, since it is not part of my metrics set)

Refactoring: To address the Long Parameter List smell in the butcher method, we can group related parameters into a new class, such as ButcherOptions. This class will encapsulate the various boolean flags and radius, making the butcher method

signature shorter and easier to understand.

Code smells from: Gonçalo Cascais 60046

Long parameter list (Reviewed by João Rivera 62877)

Code snippet:

```
public int makeShape(final Region region, final Vector3 zero, final Vector3 unit,
                     final Pattern pattern, final Expression expression, final boolean hollow, final int timeout)
        throws ExpressionException, MaxChangedBlocksException {
```

Exact location: WorldEditPrivate/worldedit-core/src/main/java/com/sk89q/worldedit /EditSession.java

Explanation of the rationale for identifying this code smell: This can make it harder to understand, maintain, and test the code. The parameters also contain related data (like zero and unit vectors, which are used to configure the environment, and parameters that define the shape and behavior like pattern, expression, hollow, and timeout).

Refactoring proposal: Group parameters like expression, pattern, and environment as ExpressionConfig. This keeps related functional parameters encapsulated and reduces the direct parameter list for makeShape.

Primitive obsession (Reviewed by Diogo Mateus 65379)

Code snippet:

```
public int makeSphere(BlockVector3 pos, Pattern block, double radiusX, double radiusY, double radiusZ, boolean filled) throws MaxChangedBlocksException {
    int affected = 0;

    radiusX += 0.5;
    radiusY += 0.5;
    radiusZ += 0.5;

    final double invRadiusX = 1 / radiusX;
    final double invRadiusY = 1 / radiusY;
    final double invRadiusZ = 1 / radiusZ;

    final int ceilRadiusX = (int) Math.ceil(radiusX);
    final int ceilRadiusY = (int) Math.ceil(radiusY);
    final int ceilRadiusZ = (int) Math.ceil(radiusZ);

    double nextXn = 0;
    forX: for (int x = 0; x <= ceilRadiusX; ++x) {
        final double xn = nextXn;
        nextXn = (x + 1) * invRadiusX;
        double nextYn = 0;
        forY: for (int y = 0; y <= ceilRadiusY; ++y) {
            final double yn = nextYn;
            nextYn = (y + 1) * invRadiusY;
            double nextZn = 0;
            forZ: for (int z = 0; z <= ceilRadiusZ; ++z) {
                final double zn = nextZn;
                nextZn = (z + 1) * invRadiusZ;
```

```
            double distanceSq = lengthSq(xn, yn, zn);
            if (distanceSq > 1) {
                if (z == 0) {
                    if (y == 0) {
                        break forX;
                    }
                    break forY;
                }
                break forZ;
            }

            if (!filled) {
                if (lengthSq(nextXn, yn, zn) <= 1 && lengthSq(xn, nextYn, zn) <= 1 && lengthSq(xn, yn, nextZn) <= 1) {
                    continue;
                }
            }

            if (setBlock(pos.add(x, y, z), block)) {
                ++affected;
            }
            if (setBlock(pos.add(-x, y, z), block)) {
                ++affected;
            }
            if (setBlock(pos.add(x, -y, z), block)) {
                ++affected;
            }
            if (setBlock(pos.add(x, y, -z), block)) {
                ++affected;
```

```
        if (setBlock(pos.add(-x, -y, z), block)) {
            ++affected;
        }
        if (setBlock(pos.add(x, -y, -z), block)) {
            ++affected;
        }
        if (setBlock(pos.add(-x, y, -z), block)) {
            ++affected;
        }
```

Exact location: WorldEditPrivate/worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java

Explanation of the rationale for identifying this code smell: The code here demonstrates a form of primitive obsession because it's relying heavily on individual x, y, and z values to represent points, rather than using a dedicated class to encapsulate this data. This leads to repeated code and potential readability issues, as calculations are performed on individual coordinates instead of through a well-defined structure.

Refactoring proposal: Create a Point3D class that stores x, y, and z as fields.

Long Method (Reviewed by João Nascimento 62896)

Code snippet:

```
public int stack(Actor actor, World world, EditSession editSession, LocalSession session,
                 @Selection Region region,
                 @Arg(desc = "# of copies to stack", def = "1")
                     int count,
                 @Arg(desc = "How far to move the contents each stack", def = Offset.FORWARD)
                 @Offset
                     BlockVector3 offset,
                 @Switch(name = 's', desc = "Shift the selection to the last stacked copy")
                     boolean moveSelection,
                 @Switch(name = 'a', desc = "Ignore air blocks")
                     boolean ignoreAirBlocks,
                 @Switch(name = 'e', desc = "Also copy entities")
                     boolean copyEntities,
                 @Switch(name = 'b', desc = "Also copy biomes")
                     boolean copyBiomes,
                 @Switch(name = 'r', desc = "Use block units")
                    boolean blockUnits,
                 @ArgFlag(name = 'm', desc = "Set the include mask, non-matching blocks become air")
                     Mask mask) throws WorldEditException {
    checkCommandArgument(count >= 1, "Count must be >= 1");

    Mask combinedMask;
    if (ignoreAirBlocks) {
        if (mask == null) {
            combinedMask = new ExistingBlockMask(editSession);
        } else {
            combinedMask = new MaskIntersection(mask, new ExistingBlockMask(editSession));
        }
    } else {
        combinedMask = mask;
    }
```

```
    int affected;
    if (blockUnits) {
        affected = editSession.stackRegionBlockUnits(region, offset, count, copyEntities, copyBiomes, combinedMask);
    } else {
        affected = editSession.stackCuboidRegion(region, offset, count, copyEntities, copyBiomes, combinedMask);
    }

    if (moveSelection) {
        try {
            final BlockVector3 size = region.getMaximumPoint().subtract(region.getMinimumPoint()).add(1, 1, 1);

            final BlockVector3 shiftSize = blockUnits ? offset : offset.multiply(size);
            final BlockVector3 shiftVector = shiftSize.multiply(count);
            region.shift(shiftVector);

            session.getRegionSelector(world).learnChanges();
            session.getRegionSelector(world).explainRegionAdjust(actor, session);
        } catch (RegionOperationException e) {
            actor.printError(e.getRichMessage());
        }
    }

    actor.printInfo(TranslatableComponent.of("worldedit.stack.changed", TextComponent.of(affected)));
    return affected;
}
```

Exact location: WorldEditPrivate/worldedit-core/src/main/java/com/sk89q/worldedit/command/RegionCommands.java

Explanation of the rationale for identifying this code smell: Long method, code readability could be improved by breaking this into smaller, focused methods.

Refactoring proposal: Simplify this logic into a separate method.

---

1 ->Code smells from: João Rivera 62877

Primitive Obsession (Reviewed by Diogo Mateus 65379)

Code snippet:

```
final int ceilRadiusX = (int) Math.ceil(radiusX);
final int ceilRadiusZ = (int) Math.ceil(radiusZ);

double nextXn = 0;
forX: for (int x = 0; x <= ceilRadiusX; ++x) {
    final double xn = nextXn;
    nextXn = (x + 1) * invRadiusX;
    double nextZn = 0;
    forZ: for (int z = 0; z <= ceilRadiusZ; ++z) {
        final double zn = nextZn;
        nextZn = (z + 1) * invRadiusZ;

        double distanceSq = lengthSq(xn, zn);
        if (distanceSq > 1) {
            if (z == 0) {
                break forX;
            }
            break forZ;
        }

        if (!filled) {
            if (lengthSq(nextXn, zn) <= 1 && lengthSq(xn, nextZn) <= 1) {
                continue;
            }
        }

        for (int y = 0; y < height; ++y) {
            if (setBlock(pos.add(x, y, z), block)) {
```

```java
public int makeCylinder(BlockVector3 pos, Pattern block, double radiusX, double radiusZ, int height, boolean filled) {
    int affected = 0;

    radiusX += 0.5;
    radiusZ += 0.5;

    if (height == 0) {
        return 0;
    } else if (height < 0) {
        height = -height;
        pos = pos.subtract( x: 0, height,  z: 0);
    }

    if (pos.y() < world.getMinY()) {
        pos = pos.withY(world.getMinY());
    } else if (pos.y() + height - 1 > world.getMaxY()) {
        height = world.getMaxY() - pos.y() + 1;
    }

    final double invRadiusX = 1 / radiusX;
    final double invRadiusZ = 1 / radiusZ;

    final int ceilRadiusX = (int) Math.ceil(radiusX);
                for (int y = 0; y < height; ++y) {
                    if (setBlock(pos.add(x, y, z), block)) {
                        ++affected;
                    }
                    if (setBlock(pos.add(-x, y, z), block)) {
                        ++affected;
                    }
                    if (setBlock(pos.add(x, y, -z), block)) {
                        ++affected;
                    }
                    if (setBlock(pos.add(-x, y, -z), block)) {
                        ++affected;
                    }
                }
            }
        }

        return affected;
    }
```

Exact Location: worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java -> method in question -> makeCylinder

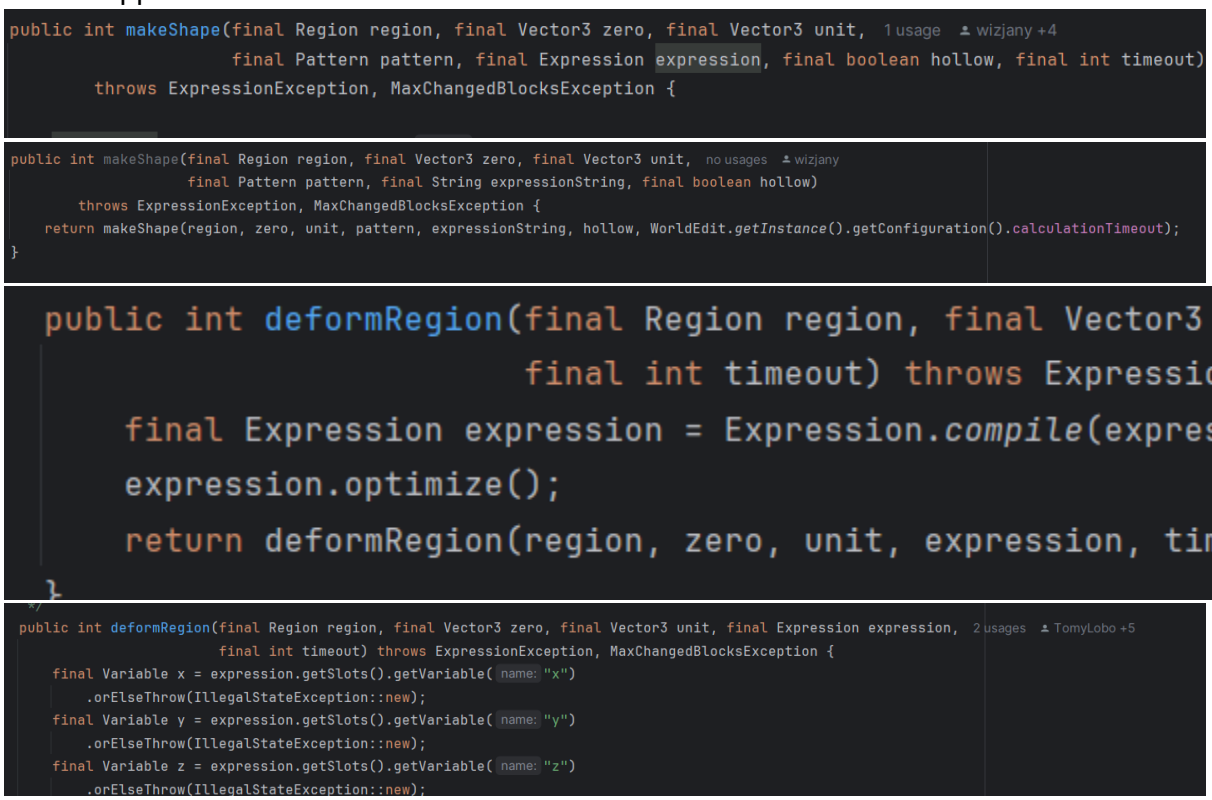
Explanation of the rationale for identifying this code smell: Was checking through the method and found a bunch of int and double usage when we are talking about coordinates.

Refactoring proposal: Create classes to store these values as points in space/plane instead of just using ints and doubles.

Note: This code smell extends to most of the makeX methods in this same class

2 -> Data Clump(Reviewed by Dimitrios Schoinas 65313)

Code snippet:



```
public int makeShape(final Region region, final Vector3 zero, final Vector3 unit, 1 usage  wizjany +4
                     final Pattern pattern, final Expression expression, final boolean hollow, final int timeout)
        throws ExpressionException, MaxChangedBlocksException {

public int makeShape(final Region region, final Vector3 zero, final Vector3 unit, no usages  wizjany
                     final Pattern pattern, final String expressionString, final boolean hollow)
        throws ExpressionException, MaxChangedBlocksException {
    return makeShape(region, zero, unit, pattern, expressionString, hollow, WorldEdit.getInstance().getConfiguration().calculationTimeout);
}

public int deformRegion(final Region region, final Vector3 zero, fin
                        final int timeout) throws ExpressionExceptio
    final Expression expression = Expression.compile(expressionStrin
    expression.optimize();
    return deformRegion(region, zero, unit, expression, timeout);
}

public int deformRegion(final Region region, final Vector3 zero, final Vector3 unit, final Expression expression,  2 usages  TomyLobo +5
                        final int timeout) throws ExpressionException, MaxChangedBlocksException {
    final Variable x = expression.getSlots().getVariable( name: "x")
        .orElseThrow(IllegalStateException::new);
    final Variable y = expression.getSlots().getVariable( name: "y")
        .orElseThrow(IllegalStateException::new);
    final Variable z = expression.getSlots().getVariable( name: "z")
        .orElseThrow(IllegalStateException::new);
```

Exact Location: worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java -> methods in question -> deformRegion (theres like 3 of them), makeShape(also like 3 of them)

Explanation of the rationale for identifying this code smell: Was scrolling through this class after finding the previous code smell and found around 6 methods in a row using the same paramaters.

Refactoring proposal: Make this collection of parameters into a class of its own, being careful so as to not create a data class.

3 -> Data Class (Reviewed by Gonçalo Cascais 60046)

Code snippet:

```
     * @param name the name
     * @param owner the owner
     */
    protected AbstractTask(String name, @Nullable Object owner) {  1 usage  Matthew Miller
        checkNotNull(name);
        this.name = name;
        this.owner = owner;
    }

    @Override  no usages  Matthew Miller
    public UUID getUniqueId() { return uniqueId; }

    @Override  Matthew Miller
    public String getName() { return name; }

    @Nullable  Matthew Miller
    @Override
    public Object getOwner() { return owner; }

    @Override  no usages  Matthew Miller
    public Date getCreationDate() { return creationDate; }
}
```

Exact Location: worldedit-core/src/main/java/com/sk89q/worldedit/util/task/AbstractTask.java -> AbstractTask class

Explanation of rationale for identifying this code smell: Opened the class and realised there were only get methods implemented.

Refactoring proposal: This class is only extended by one other class so just put into that other class these get methods.

Code smells from: Diogo Mateus 65379

<u>Long Parameter List Code Snipet: (Reviewed by: João Nascimento 62896)</u>

```
@CommandPermissions("worldedit.generation.shape")
@Logging(ALL)
public int generate(Actor actor, LocalSession session, EditSession editSession,
                    @Selection Region region,
                    @Arg(desc = "The pattern of blocks to set")
                        Pattern pattern,
                    @Arg(desc = "Expression to test block placement locations and set block type", variable = true
                        List<String> expression,
                    @Switch(name = 'h', desc = "Generate a hollow shape")
                        boolean hollow,
                    @Switch(name = 'r', desc = "Use the game's coordinate origin")
                        boolean useRawCoords,
                    @Switch(name = 'o', desc = "Use the placement's coordinate origin")
                        boolean offset,
                    @Switch(name = 'c', desc = "Use the selection's center as origin")
                        boolean offsetCenter) throws WorldEditException {
```

Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/command/GenerationCommands.java

Explanation of the rationale for identifying this code smell: In this case, the use of this many arguments could and should be avoided. In this case, it would be more clear to read if instead of passing 4 boolean variables, it was created and object called "restrictions" that stored the variables.

Message Chain Code Snipet: (Reviewed by: Dimitrios Schoinas 65313)

```
}
else if (newState != null) {
  SideEffectSet applier = session.getSideEffectSet();
  for (SideEffect sideEffectEntry : WorldEdit.getInstance().getPlatformManager().getSupportedSideEffects()) {
      if (sideEffectEntry.isExposed()) {
          applier = applier.with(sideEffectEntry, newState);
      }
```

Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/command/GeneralCommands.java

Explanation of the rationale for identifying this code smell: This is present in many parts of the overall code. This is one of the examples that is more problematic. The issue could be avoided if the Class in question only talked to neighbour Classes.

Primitive Obsession Code Snipet: (Reviewed by: Gonçalo Cascais 60046)

```
2843        for (int i = 0; i < numErodeIterations; i++) {
2844            for (int x = 0; x <= ceilBrushSize * 2; x++) {
2845                for (int y = 0; y <= ceilBrushSize * 2; y++) {
2846                    for (int z = 0; z <= ceilBrushSize * 2; z++) {
2847                        int realX = x - ceilBrushSize;
2848                        int realY = y - ceilBrushSize;
2849                        int realZ = z - ceilBrushSize;
2850                        if (lengthSq(realX, realY, realZ) > brushSizeSq) {
2851                            continue;
2852                        }
```

Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java

Explanation of the rationale for identifying this code smell: Using 4 for loops makes the code complicated and hard to read. The cause of this problem is primitive obsession. The cooder is using a loop for each axis (x,y,z). Instead of doing that, it would be much easier if he used and object that could be called "Point". This could store the information needed and it would only take a while loop to run across all combinations of x, y and z (Starting from (0,0,0) and ending in (x,y,z)).

Code smells from: João Nascimento 62896

Long Parameter List (Reviewed by João Rivera 62877)

Code snipet:

```java
/**
 * Set the values, all constructors uses this function.
 *
 * @param loc location of the view
 * @param rotationX the X rotation
 * @param rotationY the Y rotation
 * @param maxDistance how far it checks for blocks
 * @param viewHeight where the view is positioned in y-axis
 * @param checkDistance how often to check for blocks, the smaller the more precise
 */
private void setValues(Vector3 loc, double rotationX, double rotationY, int maxDistance, double viewHeight, double checkDistance) {
    this.maxDistance = maxDistance;
    this.checkDistance = checkDistance;
    this.curDistance = 0;
    rotationX = (rotationX + 90) % 360;
    rotationY *= -1;

    double h = (checkDistance * Math.cos(Math.toRadians(rotationY)));

    offset = Vector3.at((h * Math.cos(Math.toRadians(rotationX))),
                        (checkDistance * Math.sin(Math.toRadians(rotationY))),
                        (h * Math.sin(Math.toRadians(rotationX))));

    targetPosDouble = loc.add(0, viewHeight, 0);
    targetPos = targetPosDouble.toBlockPoint();
    prevPos = targetPos;
}
```

Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/util/TargetBlock.java

Explanation of the rationale for identifying this code smell: The use of a long parameter list makes it harder to understand, maintain, and test the code. A large number of parameters, especially when they are related, can obscure the function's intent and increase the cognitive load required to understand it. In this case, the parameters seem to contain both data that configures the environment (e.g., location vectors) and data that controls behavior (e.g., distance, rotation, view height).

Speculative Generality (Reviewed by Dimitros Schoinas 65313)

Code snipet:

```java
/**
 * Thrown when a maximum radius for a brush is reached.
 */
public class MaxBrushRadiusException extends MaxRadiusException {

}
```

Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/MaxBrushRadiusException.java

Explanation of the rationale for identifying this code smell: This subclass is completely empty, just extends MaxRadiusException class but doesn't add any behavior. This adds unnecessary complexity to the code.

Long Method

Code snipet:

```
public void runScript(Player player, File f, String[] args) throws WorldEditException {
    String filename = f.getPath();
    int index = filename.lastIndexOf('.');
    String ext = filename.substring(index + 1);

    if (!ext.equalsIgnoreCase("js")) {
        player.printError(TranslatableComponent.of("worldedit.script.unsupported"));
        return;
    }

    String script;

    try {
        InputStream file;

        if (!f.exists()) {
            file = WorldEdit.class.getResourceAsStream("craftscripts/" + filename);

            if (file == null) {
                player.printError(TranslatableComponent.of("worldedit.script.file-not-found", TextComponent.of(fil
                return;
            }
        } else {
            file = new FileInputStream(f);
        }

        DataInputStream in = new DataInputStream(file);
        byte[] data = new byte[in.available()];
        in.readFully(data);
        in.close();
```

```
} catch (IOException e) {
    player.printError(TranslatableComponent.of("worldedit.script.read-error", TextComponent.of(e.getMessage())));
    return;
}

LocalSession session = getSessionManager().get(player);
CraftScriptContext scriptContext = new CraftScriptContext(this, getPlatformManager().queryCapability(Capability.USER_COMMANDS),
        getConfiguration(), session, player, args);

CraftScriptEngine engine;

try {
    engine = new RhinoCraftScriptEngine();
} catch (NoClassDefFoundError ignored) {
    player.printError(TranslatableComponent.of("worldedit.script.no-script-engine"));
    return;
}

engine.setTimeLimit(getConfiguration().scriptTimeout);

Map<String, Object> vars = new HashMap<>();
vars.put("argv", args);
vars.put("context", scriptContext);
vars.put("player", player);

try {
    engine.evaluate(script, filename, vars);
} catch (ScriptException e) {
    // non-exceptional return check
    if (!(Throwables.getRootCause(e) instanceof ReturnException)) {
        player.printError(TranslatableComponent.of("worldedit.script.failed", TextComponent.of(e.getMessage(), TextColor.WHITE)
```

```
            logger.warn("Failed to execute script", e);
        }
    } catch (NumberFormatException | WorldEditException e) {
        throw e;
    } catch (Throwable e) {
        player.printError(TranslatableComponent.of("worldedit.script.failed-console", TextComponent.of(e.getClass().getCanonicalNam
                TextColor.WHITE)));
        logger.warn("Failed to execute script", e);
    } finally {
        for (EditSession editSession : scriptContext.getEditSessions()) {
            editSession.close();
            session.remember(editSession);
        }
    }
}
```

Exact location: worldedit-core/src/main/java/com/sk89q/worldedit/WorldEdit.java

Explanation of the rationale for identifying this code smell: This method is too long and may be hard to maintain, hard to test, and most importantly, hard to read. Refactoring this into smaller methods could help with that for the future.

4. Use Case Diagrams

Use cases from: Dimitrios Schoinas 65313 (Reviewed by João Rivera 62877)

/tool tree

Use Case Name: Bind Tree generator tool

Description: Allows the user to bind a tool that places tree structures.

Primary Actor: Player

Secondary Actor: None

/tool deltree

Use Case Name: Bind Floating tree remover tool

Description: Allows the user to bind a tool that deletes trees or similar structures.

Primary Actor: Player

Secondary Actor: None

/tool floodfill

Use Case Name: Bind Flood fill tool

Description: Allows the user to bind a tool that fills an area with a specified material.

Primary Actor: Player

Secondary Actor: None

/tool none

Use Case Name: Clear Tool Selection

Description: Clears any active tool selection, disabling current tool actions.

Primary Actor: Player

Secondary Actor: None

/tool selwand

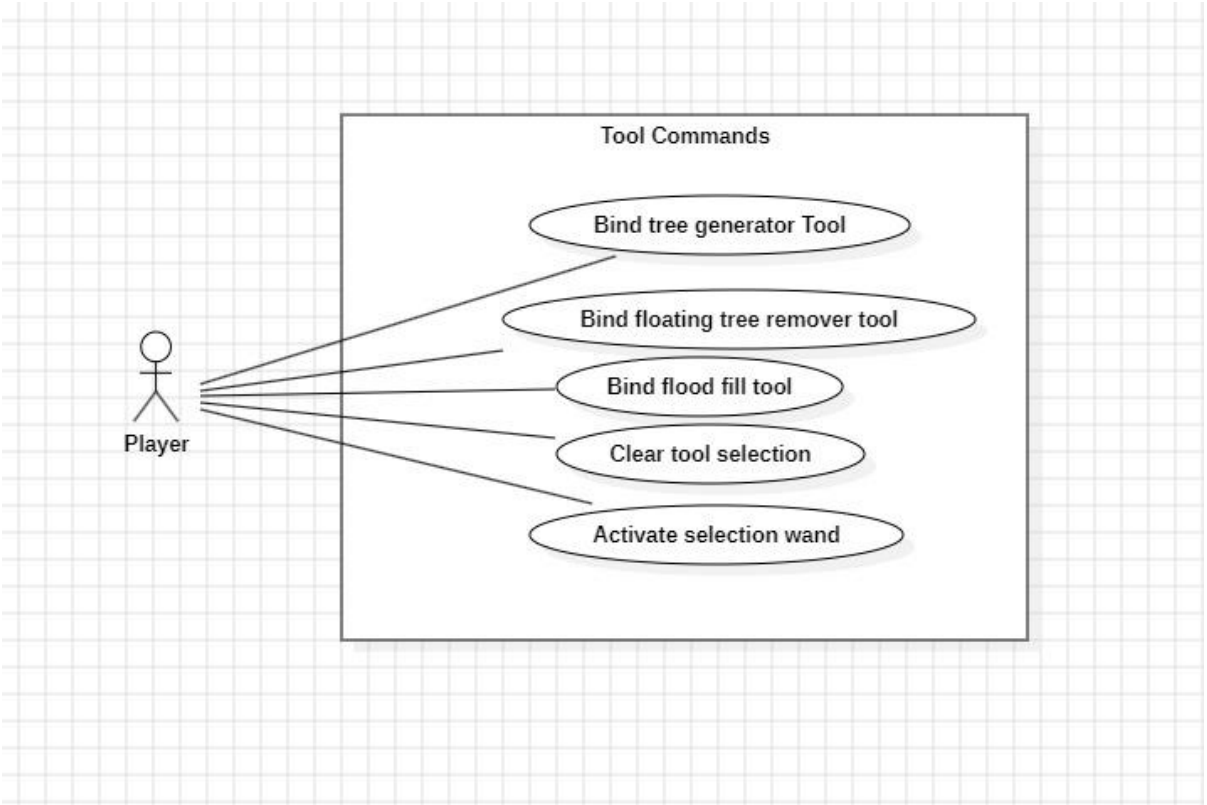
Use Case Name: Activate Selection Wand

Description: Activates the selection wand tool, allowing users to mark areas for editing.

Primary Actor: Player

Secondary Actor: None

Use case diagram:



Use Cases from João Rivera 62877 (Reviewed By Dimitrios Schoinas 65313):

Use case name: Select position 1
Use case description: The Player executes this command using //pos1[coordinates]. The WorldEdit System processes the command and executes it, selecting the position for the first corner of the cuboid, if no argument is passed it defaults to player position.
Main actor: Player
Secondary actor: None

Use case name: Generate hollow cylinder
Use case description: The Player executes this command using //hcyl <pattern> <radii> [height]. The WorldEdit System processes the command and executes it, generating the hollow cylinder with the block, radius and height specified by the Player.
Main actor: Player
Secondary actor: None

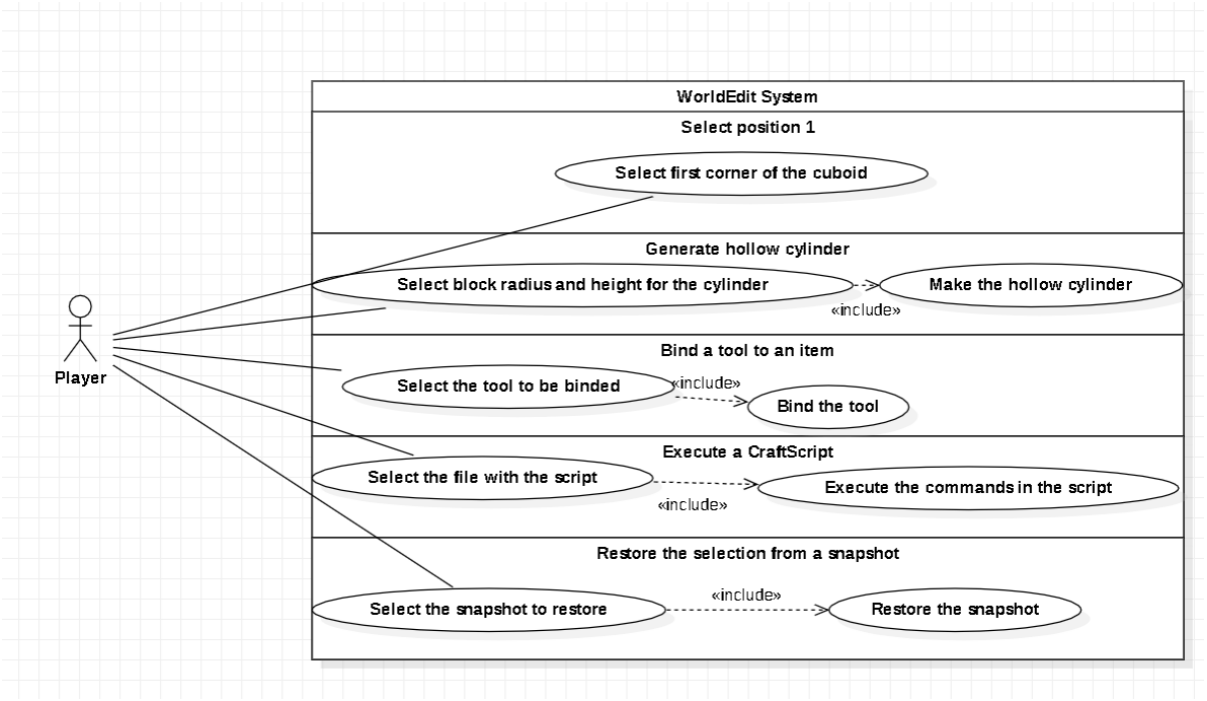
Use case name: Bind a tool to an item
Use case description: The Player executes this command using the /tool <cycler|navwand|none|floodfill|deltree|farwand|selwand|stacker|repl|tree|lrbuild|info> command. The WorldEdit System processes the command and executes it, transforming the item into the tool specified by the Player.

Main actor: Player
Secondary actor: None

Use case name: Execute a CraftScript
Use case description: The Player executes a CraftScript using the /cs <filename> [args...] command. The WorldEdit System processes the command, verifies the player's permissions, loads the specified script file, and executes it. If successful, the system performs the actions defined in the script and provides feedback to the player
Main actor: Player
Secondary actor: None

Use case name: Restore the selection from a snapshot
Use case description: The Player executes this command using the /restore [snapshot] command. The WorldEdit System processes the command and executes it, restoring the selection from the snapshot the Player provided.
Main actor: Player
Secondary actor: None

Use case diagram:



Use Cases from Gonçalo Cascais 60046 (Reviewed By Dimitrios Schoinas):

/brush forest
Use Case Name: Bind Forest generator brush
Description: Forest brush, creates a forest in the area.
Primary Actor: Player
Secondary Actor: None

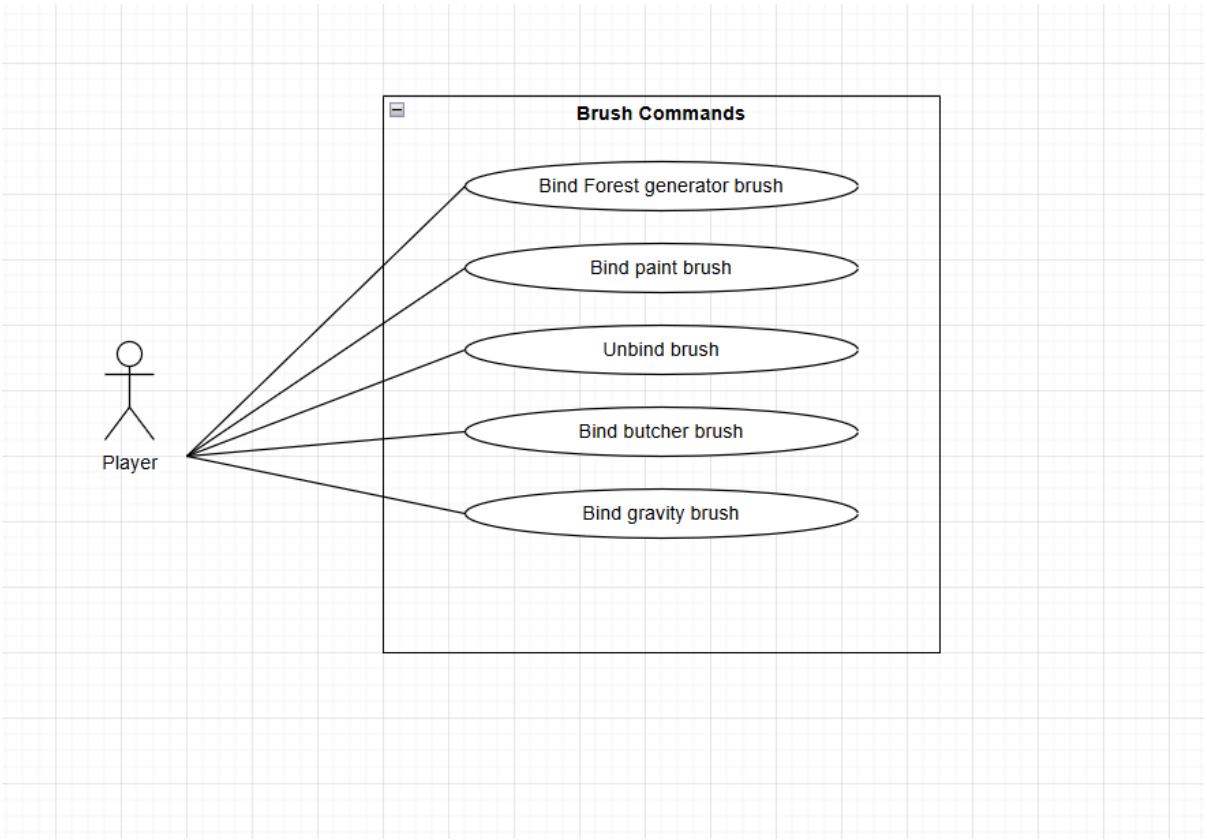
/brush butcher
Use Case Name: Bind butcher brush
Description: Butcher brush, kills mobs within a radius.
Primary Actor: Player
Secondary Actor: None

/brush paint
Use Case Name: Bind paint brush
Description: Paint brush, apply a function to a surface.
Primary Actor: Player
Secondary Actor: None

/brush none
Use Case Name: Unbind brush
Description: Unbind a bound brush from your current item
Primary Actor: Player
Secondary Actor: None

/brush gravity
Use Case Name: Bind gravity brush
Description: Gravity brush, simulates the effect of gravity
Primary Actor: Player
Secondary Actor: None

Use case diagram:


Brush Commands
Bind Forest generator brush
Bind paint brush
Unbind brush
Bind butcher brush
Bind gravity brush
Player

Use cases from: Diogo Mateus 65379 (Reviewed by: Gonçalo Cascais 60046)

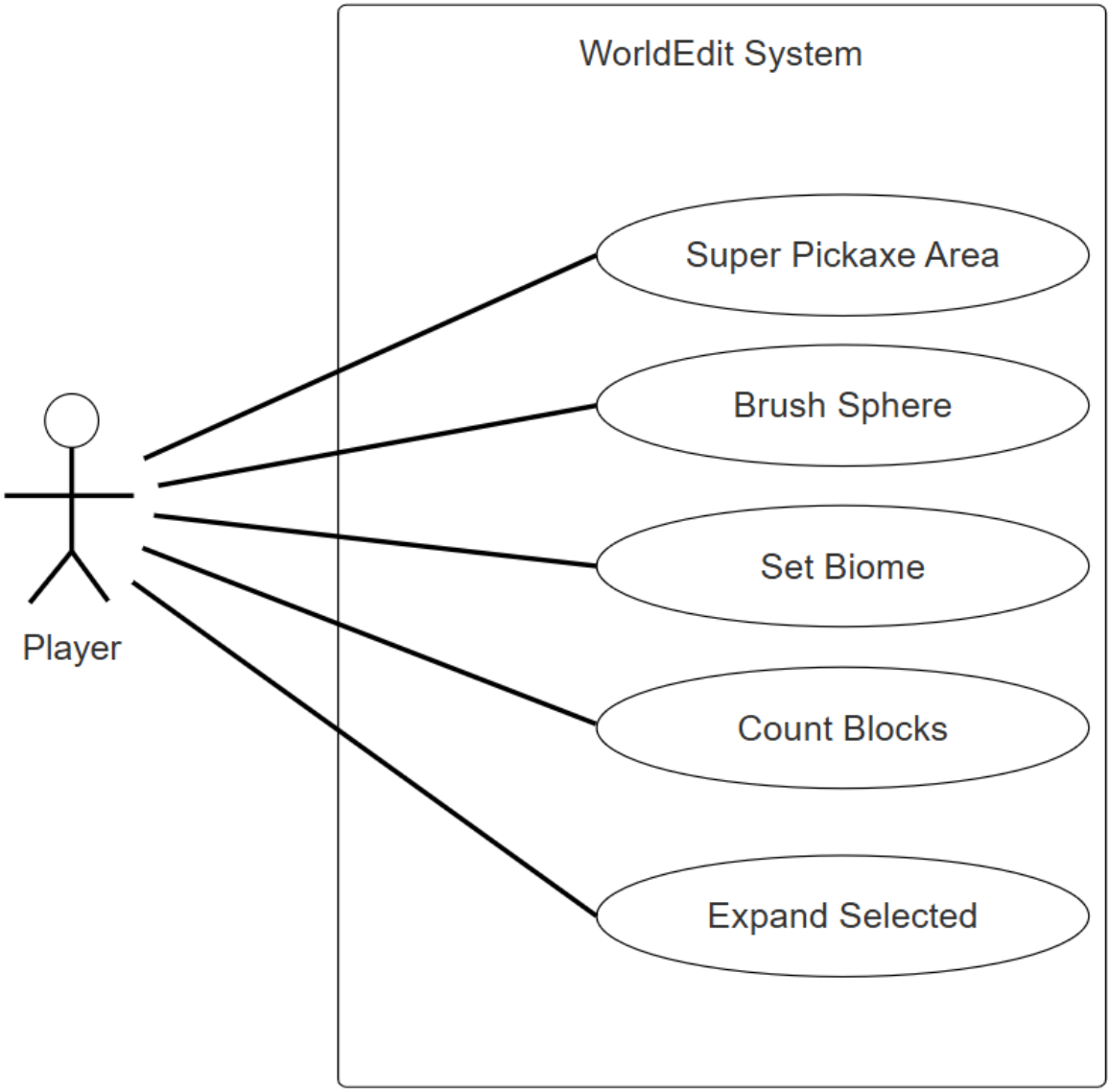
Use case name: Super Pickaxe Area / Use case description: The player executes the command: "/superpickaxe area ". This enables the Super Pickaxe area mode giving the value "range" as the range of the area pickaxe. / Main actor: Player Secondary actor: None

Use case name: Brush Sphere / Use case description: The player executes the command: "/brush sphere [-h] [radius]". This applys an effect to an item where it creates a sphere hollow or filled (indicated by [-h]), of a certain type of block () with a certain radius ([radius]) / Main actor: Player Secondary actor: None

Use case name: Set Biome / Use case description: The player executes the command: "//setbiome [-p] ". This changes the Biome to the one specified in . The [-p] is an optional argument that indicates the position of the biome to be changed. If this argument is not specified, it changes the biome the Player is currently in. / Main actor: Player Secondary actor: None

Use case name: Count Blocks / Use case description: The player executes the command: "//count ". This counts the number of blocks that match the , the being a type of block. / Main actor: Player Secondary actor: None

Use case name: Expand Selected / Use case description: The player executes the command: "//expand <vert| [reverseAmount] [direction]". This expands the selected area by an amount (), where the [reverseAmout] is the amount to expand the selection by in the other direction and [direction] is the direction where you want to expand. You can also instead of the to expand the selected are vertically until it hits the sky limit. / Main actor: Player Secondary actor: None



Use cases from: João Nascimento 62896 (Reviewed by: Diogo Mateus 65379)

/wand
Use Case Name: Get wand
Description: Gives the defined wand to select positions.
Primary Actor: Player
Secondary Actor: None

/hpos1
Use Case Name: Select Position 1 from Targeted Block
Description: Set position 1 to the block being targeted.
Primary Actor: Player
Secondary Actor: None

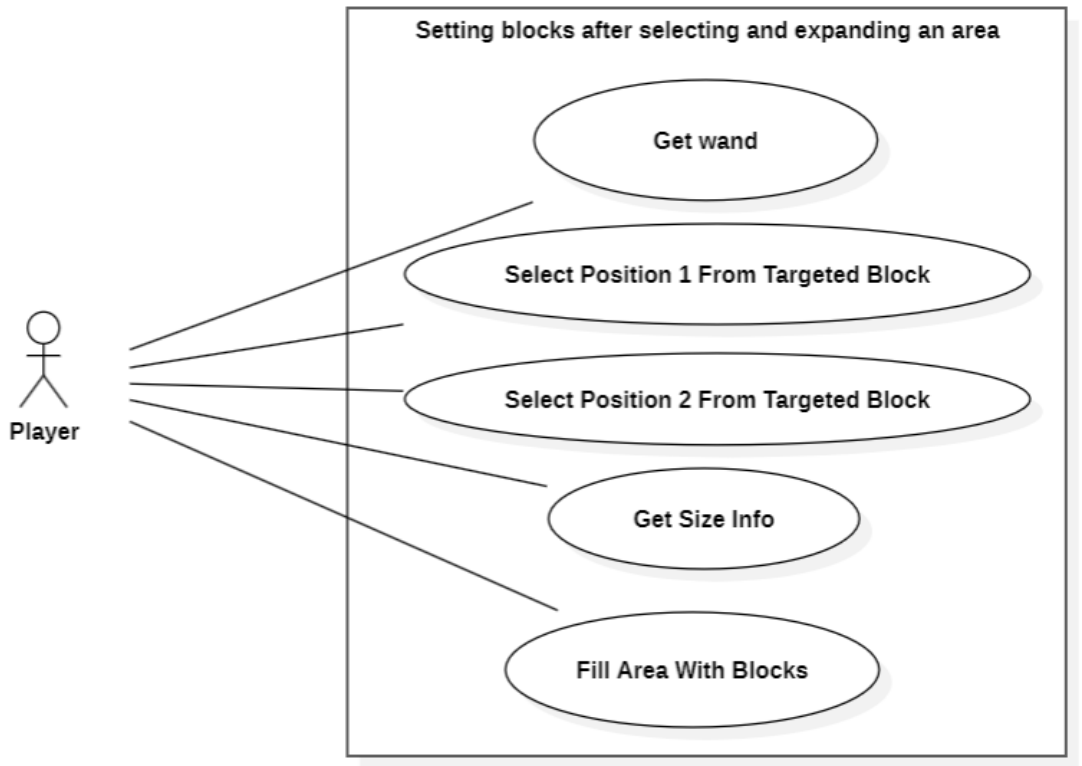
/hpos2
Use Case Name: Select Position 2 from Targeted Block
Description: Set position 2 to the block being targeted.

Primary Actor: Player
Secondary Actor: None

/size
Use Case Name: Get Size info
Description: Gives information about the size of the selection.
Primary Actor: Player
Secondary Actor: None

/set (pattern)
Use Case Name: Fill Area with Blocks
Description: Fills the area with the blocks of a given pattern
Primary Actor: Player
Secondary Actor: None

Use case diagram:



M I L E S T O N E 3

Updated user stories:

[Clipboard feature] [João Nascimento 62896]

As a Minecraft Town Builder, I want a clipboard feature so that I can quickly access and organize multiple schematics at once, helping me finding my projects faster and finish tasks more efficiently and aligning seamlessly with the overall mod experience.

[Dynamic Weather Effects on Builds] [Dimitrios Schoinas 65313]

As a player I want a way to apply visual weather effects, such as rain, snow, drought and hell ,to specific areas of the building so that it is easier to change the aspect of my buildings depending on the weather.

Note These effects would simulate the impact of weather on the blocks, altering their appearance to reflect wet, frozen or weathered conditions so that I can create a visually immersive experience.

[Artificial Lake creator] [Gonçalo Cascais 60046]

As a Minecraft player I want to be able to create an on-demand artificial lake to fill in the space around my house so that I can give it an amazing view and a feel of nature.

Implemented commands:

//schem favorite <schematic name>

Use case: Mark favorite

Description: Mark a schematic as favorite

Main actor: Player

Secondary actor: None

Pre-conditions:

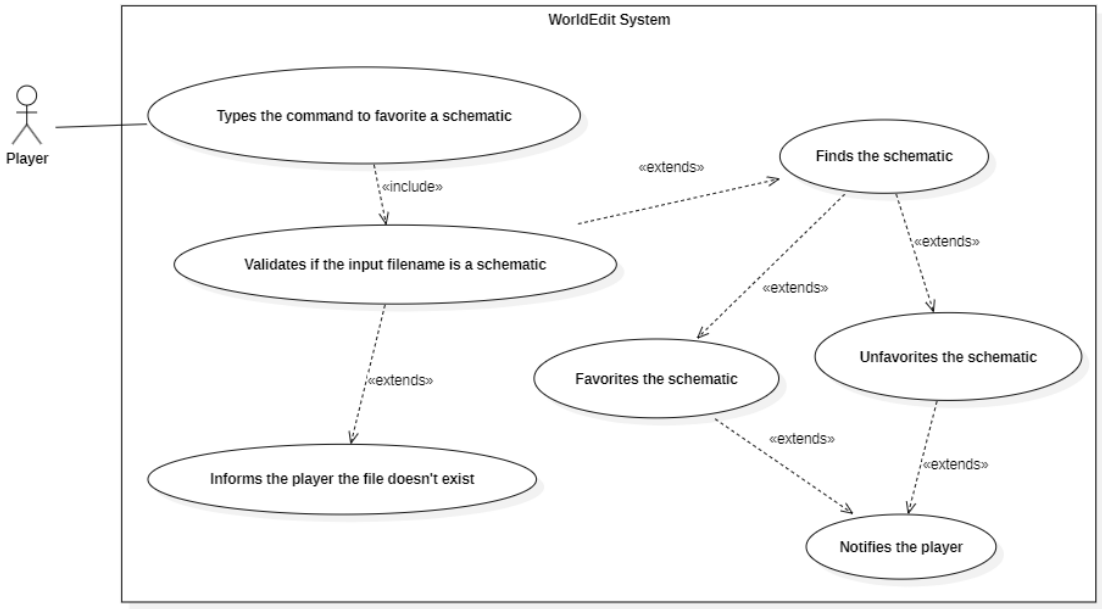
1. The player is a valid actor.
2. The schematic selected exists and is valid.
3. The schematic directory exists.
4. The Favorite Manager class is configured.

Main flow:
1. The player executes the command //schem favorite with a schematic name (//schematic favorite <schematic name>).
2. Gets the schematic directory.
3. Validates the schematic file with the name given and if it's inside a safe directory.
If not, an error message is sent to the player.
4. The system retrieves the favorite schematics for the player using the Favorite Manager class.
5. Checks if the schematic is already favorited.
If it is, removes from favorite.
If not, adds to the favorites.
6. Sends the appropriate message to the player, based on if the schematic is added or removed from favorites.

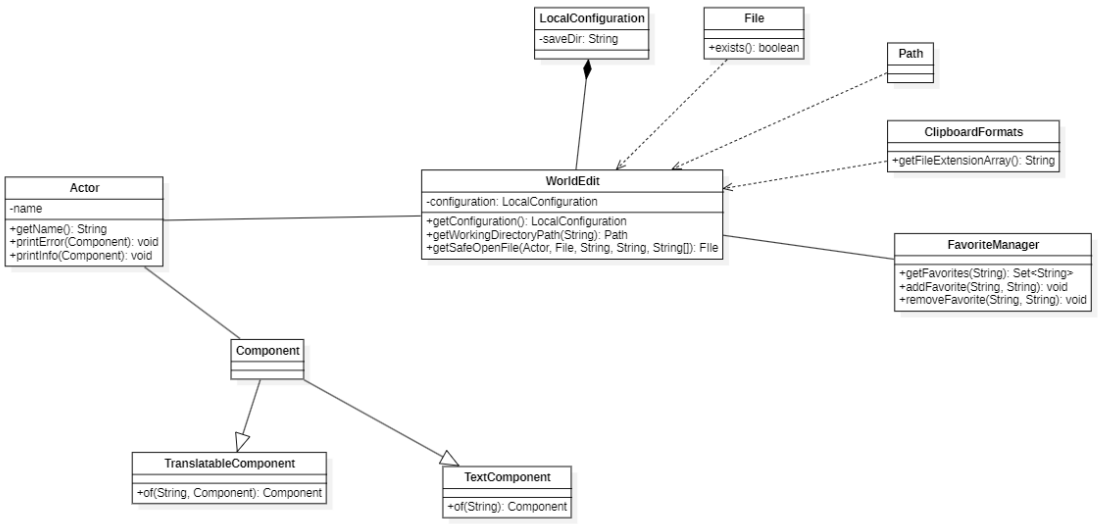
Post conditions: The schematic is added to the favorites file.

Alternate flows:
1. File not found:
If the name provided isn't a schematic name, gives an error message ("Schematic [schematic-name] does not exist!").

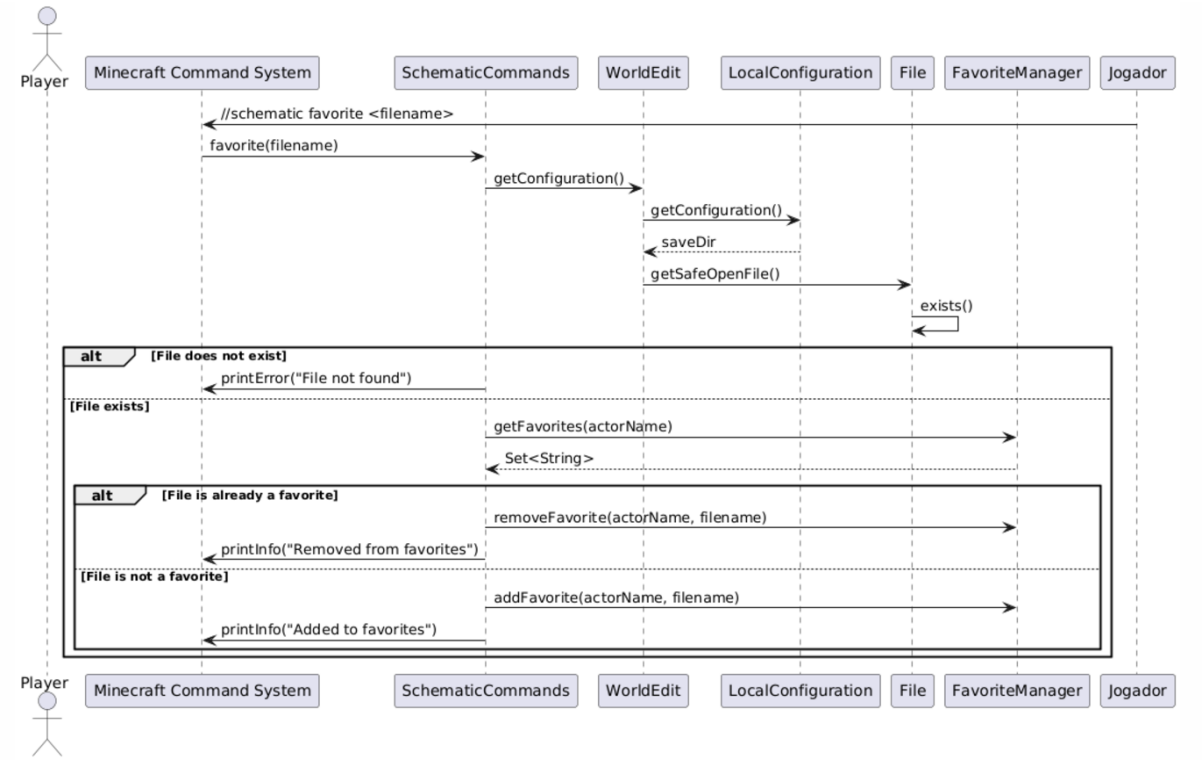
Use cases



Case diagram



Sequence diagram



Scenario based testing:

Pre-condition: Logged in
Test case id: Favorite_Schematic_OK

Steps:

1. Download the schematic: finalProject.schem (available on our GitHub repository). (If already downloaded, skip this step.)

2. Place the finalProject.schem file in the schematics folder. (Path: worldedit-fabric\run\config\worldedit\schematics)

3. In Minecraft, type: //schematic favorite example

Expected Results:
A file named finalProject.schem exists in the schematics folder on your computer.
Adding the file to the folder does not produce any errors.
After typing the command in the chat, the following scenarios occur:
If the schematic is not already in the favorites list:
A message appears in the chat: "Schematic 'finalProject' added to favorites."
The schematic name is added to the user's favorites list.
If the schematic is already in the favorites list:
A message appears in the chat: "Schematic 'finalProject' removed from favorites."

The schematic name is removed from the user's favorites list.
No unexpected errors or crashes occur during the execution of the command.

Tour report:

To implement this command we created a method in SchematicCommands Class(1) favorite() (line 360). This method is registered as a WorldEdit command using the `@Command` annotation and allows users to add or remove schematics from their favorites list. It begins by retrieving the configuration from the **WorldEdit Class (2)** via the `getConfiguration()` method to determine the directory where schematics are saved. Using the `getSafeOpenFile()` method from the same class, the command ensures the schematic file exists and is valid. If the file is missing, the method immediately notifies the user with an error message using the **TranslatableComponent (5)** class.

Next, the method interacts with the **FavoriteManager Class (3)** to retrieve the list of a user's favorites. If the schematic is already in the list, it is removed using the `removeFavorite()` method; otherwise, it is added via `addFavorite()`. After each operation, appropriate feedback is displayed to the user through Minecraft chat using the **TextComponent and TranslatableComponent classes. To support localization, new** error and success messages, such as "`worldedit.schematic.favorite.does-not-exist`", were added to the `strings.json` file in the **resources/lang (4)** directory. This ensures that the command integrates seamlessly with the existing WorldEdit structure.

Paths to classes mentioned:
(1)worldedit-core/src/main/java/com/sk89q/worldedit/command/SchematicCommands.java
WorldEdit.java
(2)worldedit-core/src/main/java/com/sk89q/worldedit/WorldEdit.java
(3)worldedit-core/src/main/java/com/sk89q/worldedit/command/util/FavoriteManager.java
strings.json
(4)worldedit-core/src/main/resources/lang/strings.json

//schem search <name>

Use case: Lists schematics with the name

Description: Searches the schematics that contains the input name on their name

Main actor: Player

Secondary actor: None

Pre-conditions:
1. The player is a valid actor.

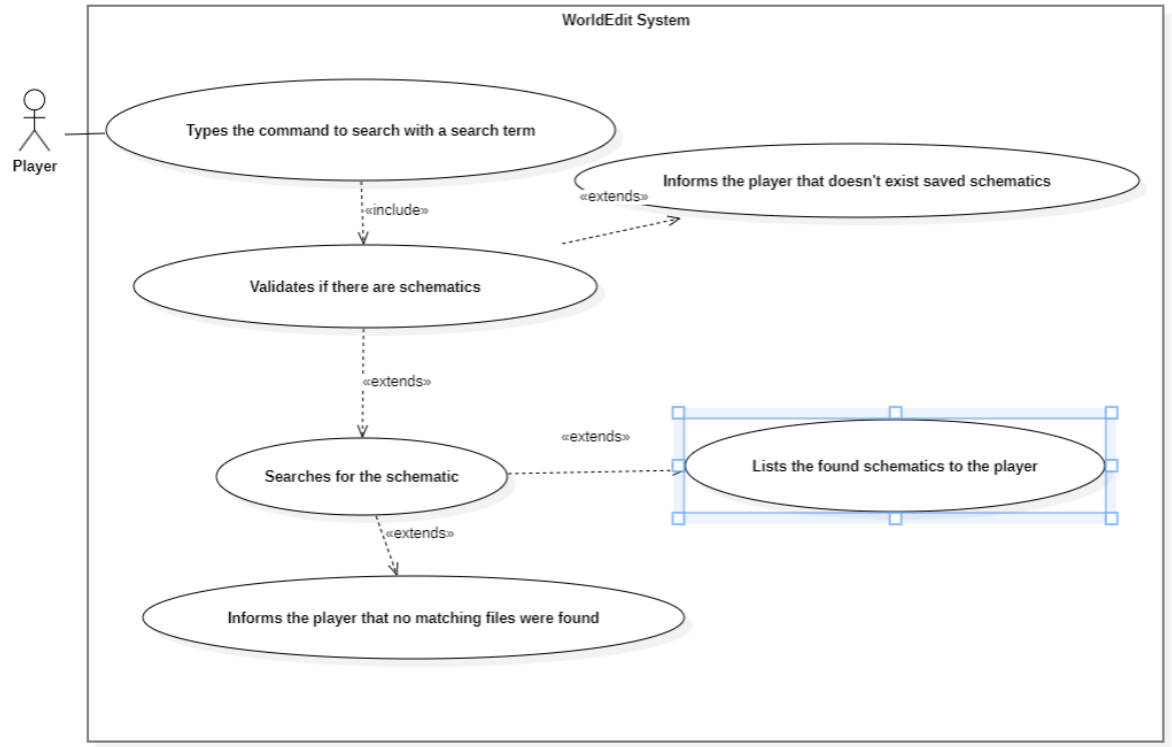
Main flow:

1. The player executes the command //schem search with a search term (//schematic search <search term>).
2. Gets the schematic directory.
3. Iterates through all files in the schematics directory, filtering files based on the search term. It is case-insensitive and matches substing in the file name.
4. The system lists every schematic found for the player in an alphabetical order, using the SchematicPaginationBox class.
The paginated results are sent in player's chat, having a clickable [L] link to load the schematic, the name of the schematic file with the format, and an additional hover text.

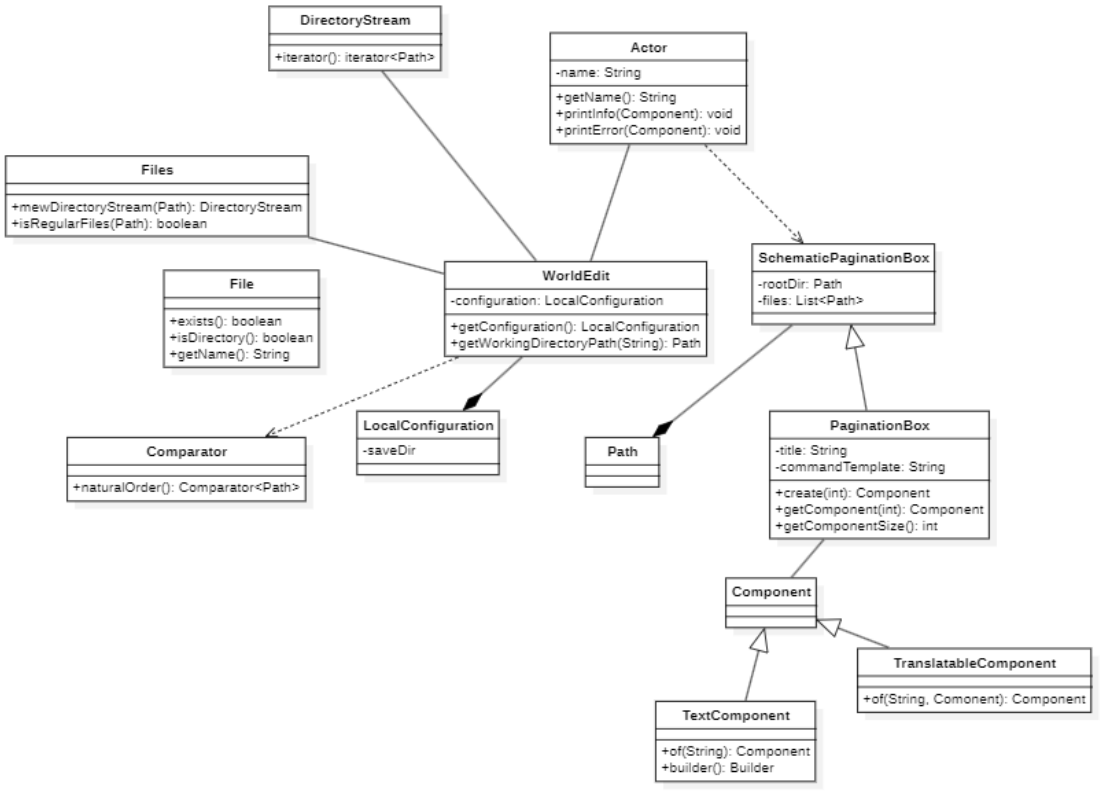
Post conditions: None.

Alternate flows:
1. No matching Files Found:
If no schematics match the search term, an error message is displayed: ("Schematic [schematic-name] does not exist!").
2. Non Existant Schematics:
If there are no schematics saved, this error message is displayed:
("There are no Schematics saved.")

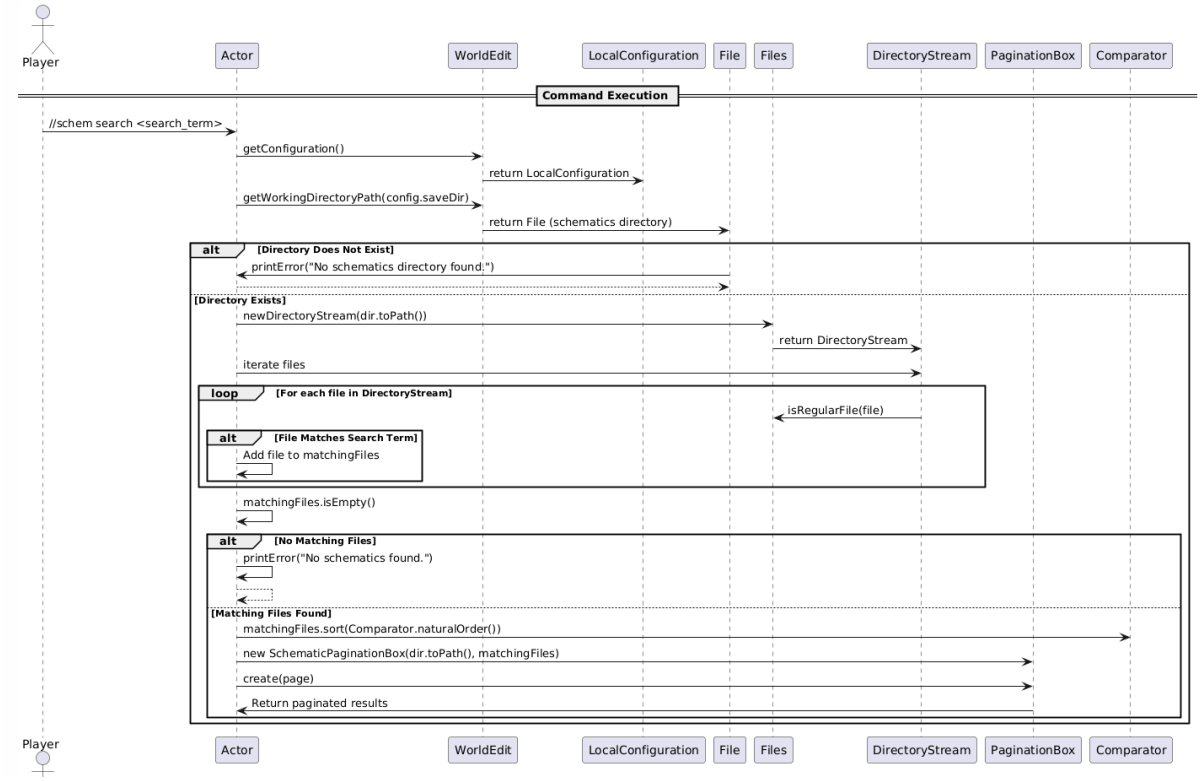
Use cases



Class diagram



Sequence diagram



Scenario based testing

Pre condition: logged in
Test case id: Search_functionality ok
Steps:
1. Navigate to the folder at worldedit-fabric\run\config\worldedit\schematics.
2. Ensure the folder contains multiple schematic files, including at least one with "house" in its name (e.g., modern_house.schem).
3. In Minecraft, type the command://schematic search house
4. Observe the chat window for a paginated list of results matching the term"house."
5. To navigate to the next page, type: //schematic search house -p 2
Expected results:
1. The schematics folder exists and contains files for the test.
2. The command //schematic search house does not return errors.
3. The chat displays a paginated list of matching schematic names, starting with the first page.
4. The pagination system works correctly when using the -p flag for additional pages.

Tour report:

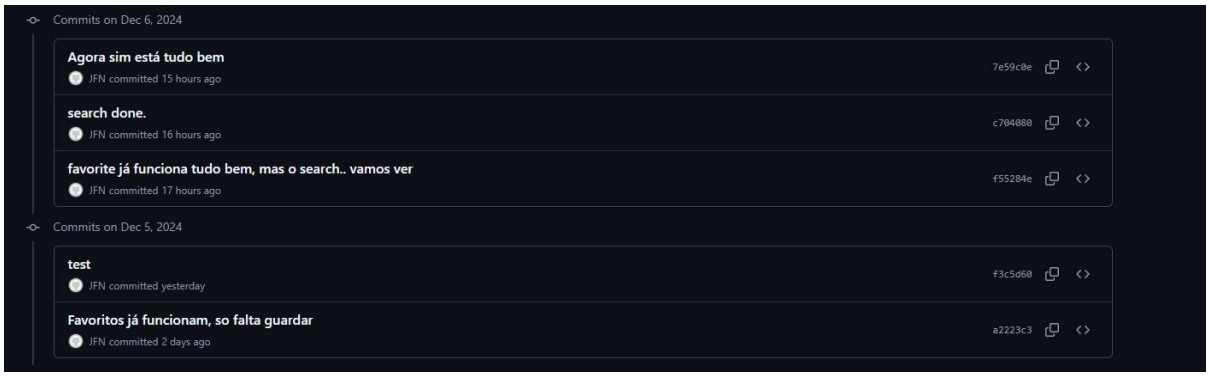
To implement the search command, we added a method called search() in the SchematicCommands (1) class. This method begins by accessing the schematics folder, whose path is retrieved from the LocalConfiguration object. It verifies the folder's existence and then iterates through its contents using a DirectoryStream. Each file name is normalized to lowercase for case-insensitive matching against the user-provided search term. Matching file paths are added to a list for processing.
To handle the display of results, we introduced the SchematicPaginationBox (2) class, which extends the existing PaginationBox (3) class. This specialized class builds a paginated list of schematics, including clickable links (e.g., [L]) that allow players to load schematics directly via the chat interface.
Sorting of files is managed using Comparator.naturalOrder(), ensuring that results are displayed in alphabetical order. Error messages for various failure cases (e.g., empty folder, no matches, invalid pages) are defined using TranslatableComponent and localized string templates.

(1) worldedit-core/src/main/java/com/sk89q/worldedit/command/SchematicCommands.java

(2) worldedit-core/src/main/java/com/sk89q/worldedit/command/SchematicCommands.java

(3) worldedit-core/src/main/java/com/sk89q/worldedit/util/formatting/component/PaginationBox.java

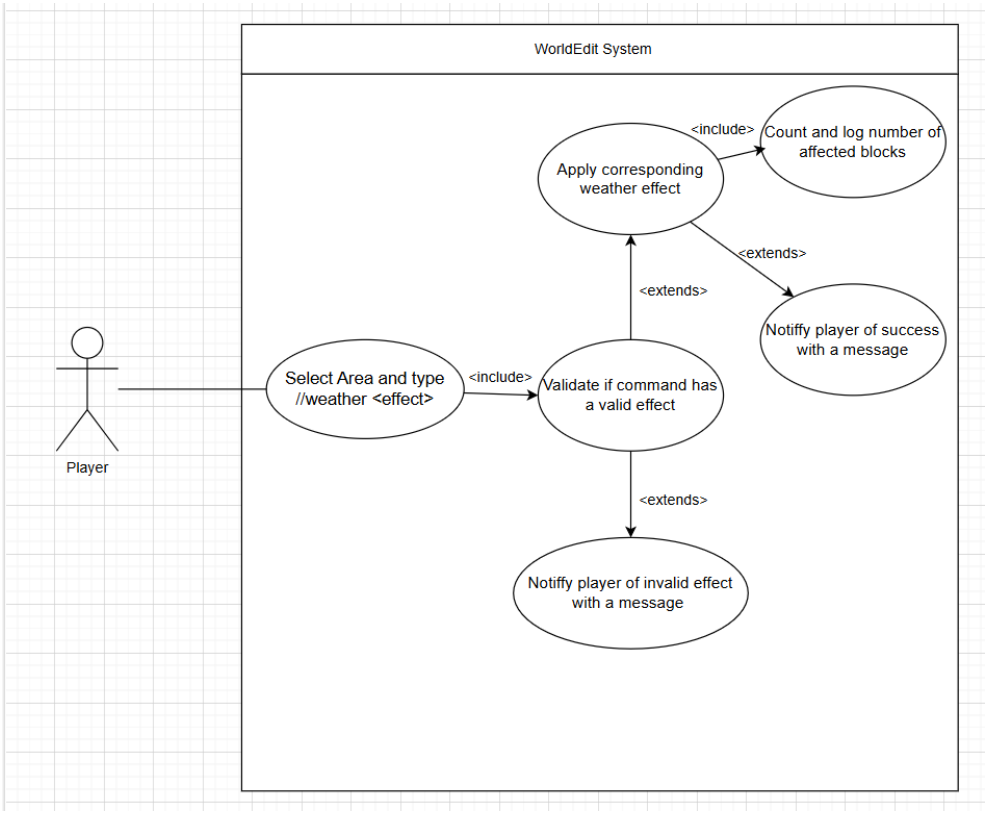
The code was all made by João Nascimento, both the favorites and the search:



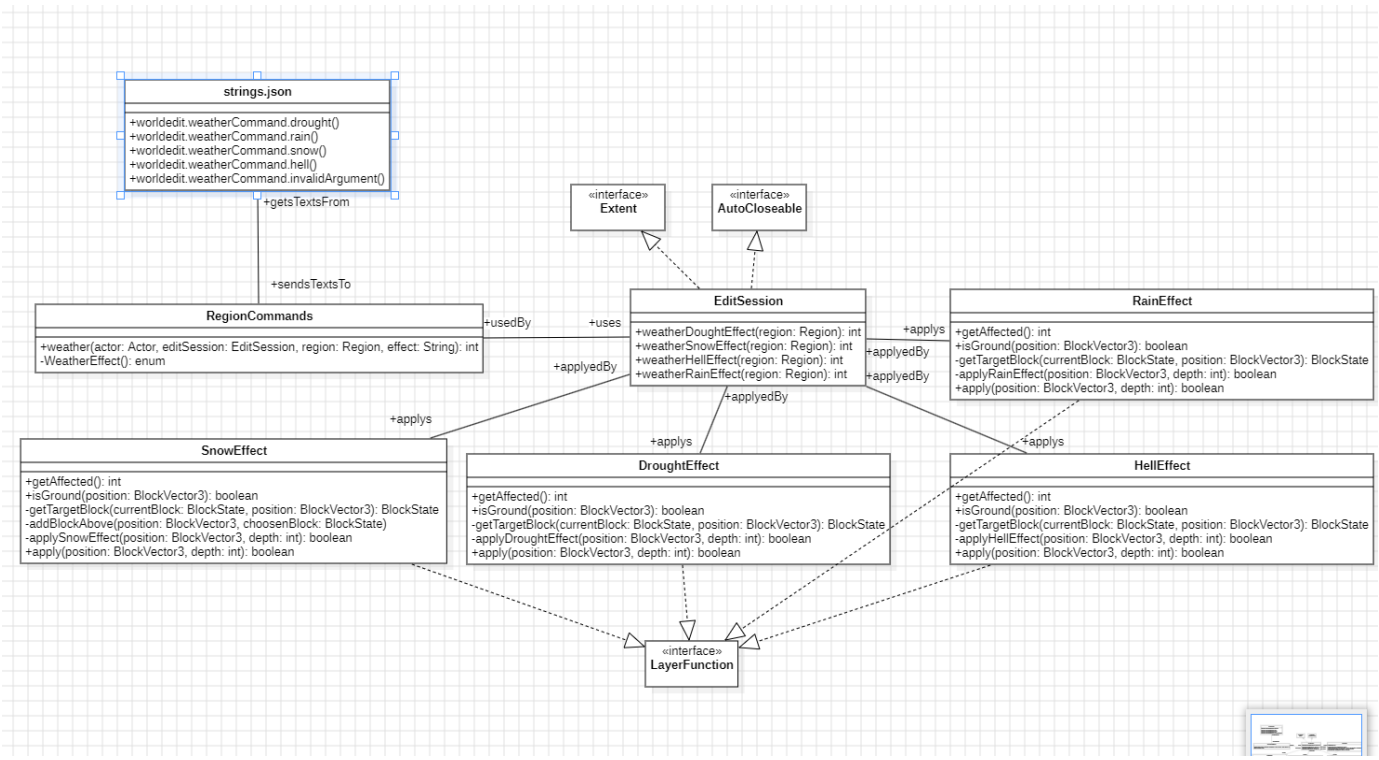
//weather <argument> (reviewers: Dimitrios and Diogo)

Use case: Apply weather effect
Id: 1
Description: Transforms the blocks in the selected region to reflect the chosen weather condition (drought, rain, snow, hell).
Main actor: Player
Secondary actor: None
Pre-conditions: 1.The actor is a valid actor.
2.The editSession is the current session we are working in.
3. There is a selected region.
Main flow: 1. The player executes the //weather command with a specified effect (//weather <effect>).
2.The system validates the provided effect.
If valid, the corresponding weather effect is applied to the region using the editSession.
If invalid, the system notifies the player with an error message.
3.The system counts and logs the number of affected blocks.
4.A success message is displayed to the player.
Post conditions: The blocks in the selected region are modified to reflect the chosen weather condition.
A message confirming the operation or indicating an error is displayed to the player.
Alternative flows:
1.Invalid Effect Name:
If the provided effect name does not match a valid weather effect, the system displays an error message (“You should choose between [drought | snow | rain | hell]”). No changes are applied.

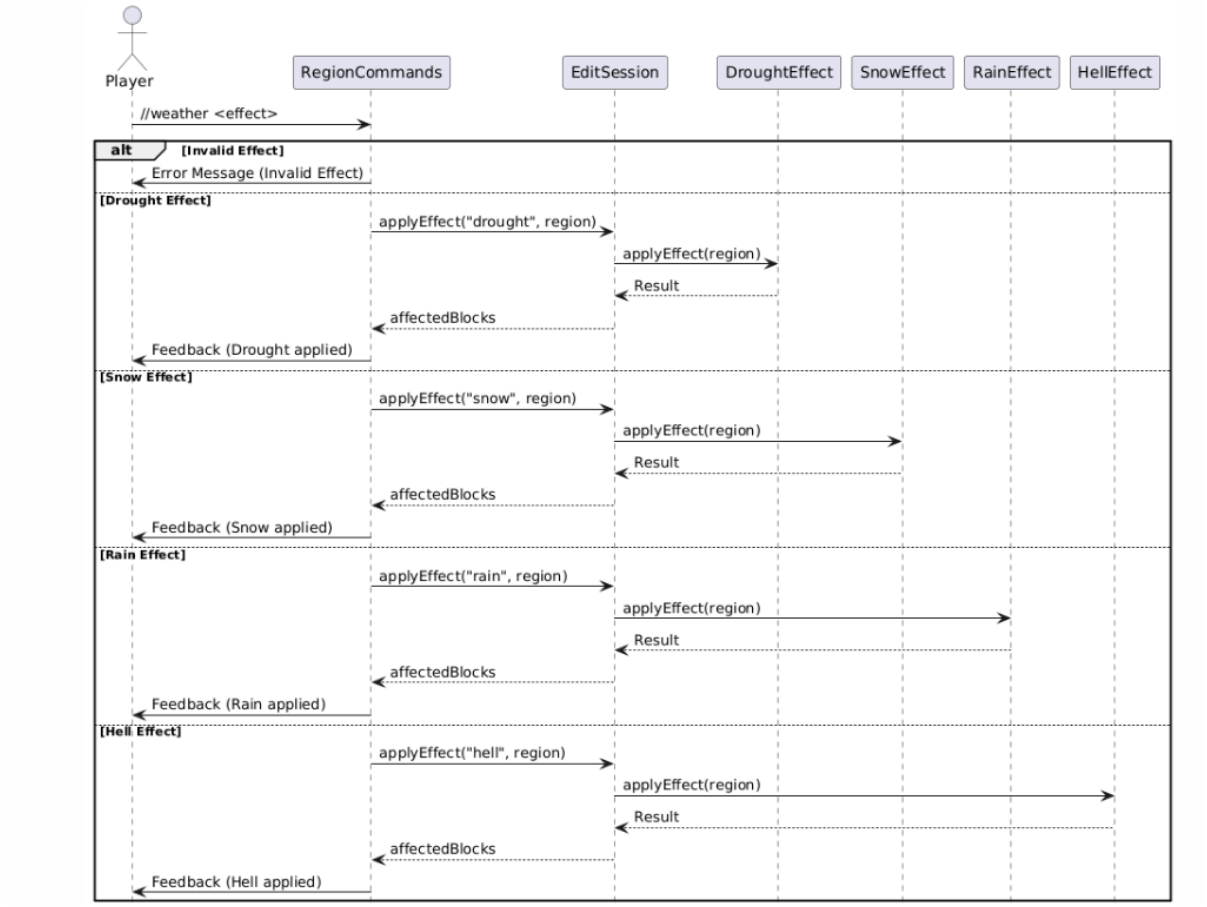
Use Cases



Class Diagram



Sequence diagram



Scenario Based Testing:

- **Drought:**

Pre-condition: Logged in
Test case id: Drought Effect ok
Steps: 1. Download the schematic : “finalProject.schem” (available on our gitHub repository) (if already downloaded ignore this step)
2. Put the “finalProject.schem” file in the schematics folder (the path is: worldedit-fabric\run\config\worldedit\schematics)
3. In minecraft type: “//schematic load finalProject schem”
4. In minecraft type: “//paste”
5. With your Wooden Axe left click the leave block situated on the top left corner of the loaded building
6. With your Wooden Axe right click the yellow concrete block situated on the bottom right corner of the loaded building
7. In minecraft type: //weather drought
Expected results: 1. You should have a file on your computer named “finalProject.schem”
2. Is expected to have no errors when adding the file to the folder
3. In chat should appear a message saying: “finalProject loaded. Paste it with //paste”
4. The building is generated near the player
5. In chat should appear a message saying: “First position set to (x, y ,z) (n)”

6. In chat should appear a message saying: “Second position set to (x, y ,z) (n)”
7. The selected area changes and is printed a message saying “Transformed the blocks to simulate a drought environment!”

- **Snow:**

Pre-condition: Logged in
Test case id: Snow Effect ok
Steps: 1. Download the schematic : “finalProject.schem” (available on our gitHub repository) (if already downloaded ignore this step)
2. Put the “finalProject.schem” file in the schematics folder (the path is: worldedit-fabric\run\config\worldedit\schematics)
3. In minecraft type: “//schematic load finalProject schem”
4. In minecraft type: “//paste”
5. With your Wooden Axe left click the leave block situated on the top left corner of the loaded building
6. With your Wooden Axe right click the yellow concrete block situated on the bottom right corner of the loaded building
7. In minecraft type: //weather snow
Expected results: 1. You should have a file on your computer named “finalProject.schem”
2. Is expected to have no errors when adding the file to the folder
3. In chat should appear a message saying: “finalProject loaded. Paste it with //paste”
4. The building is generated near the player
5. In chat should appear a message saying: “First position set to (x, y ,z) (n)”
6. In chat should appear a message saying: “Second position set to (x, y ,z) (n)”
7. The selected area changes and is printed a message saying “Transformed the blocks to simulate a snowy environment!”

- **Hell:**

Pre-condition: Logged in
Test case id: Hell Effect ok
Steps: 1. Download the schematic : “finalProject.schem” (available on our gitHub repository) (if already downloaded ignore this step)
2. Put the “finalProject.schem” file in the schematics folder (the path is: worldedit-fabric\run\config\worldedit\schematics)
3. In minecraft type: “//schematic load finalProject schem”
4. In minecraft type: “//paste”
5. With your Wooden Axe left click the leave block situated on the top left corner of the loaded building
6. With your Wooden Axe right click the yellow concrete block situated on the bottom right corner of the loaded building
7. In minecraft type: //weather hell
Expected results: 1. You should have a file on your computer named “finalProject.schem”
2. Is expected to have no errors when adding the file to the folder
3. In chat should appear a message saying: “finalProject loaded. Paste it with //paste”
4. The building is generated near the player
5. In chat should appear a message saying: “First position set to (x, y ,z) (n)”
6. In chat should appear a message saying: “Second position set to (x, y ,z) (n)”

7. The selected area changes and is printed a message saying “Transformed the blocks to simulate a hell environment!”

- **Rain:**

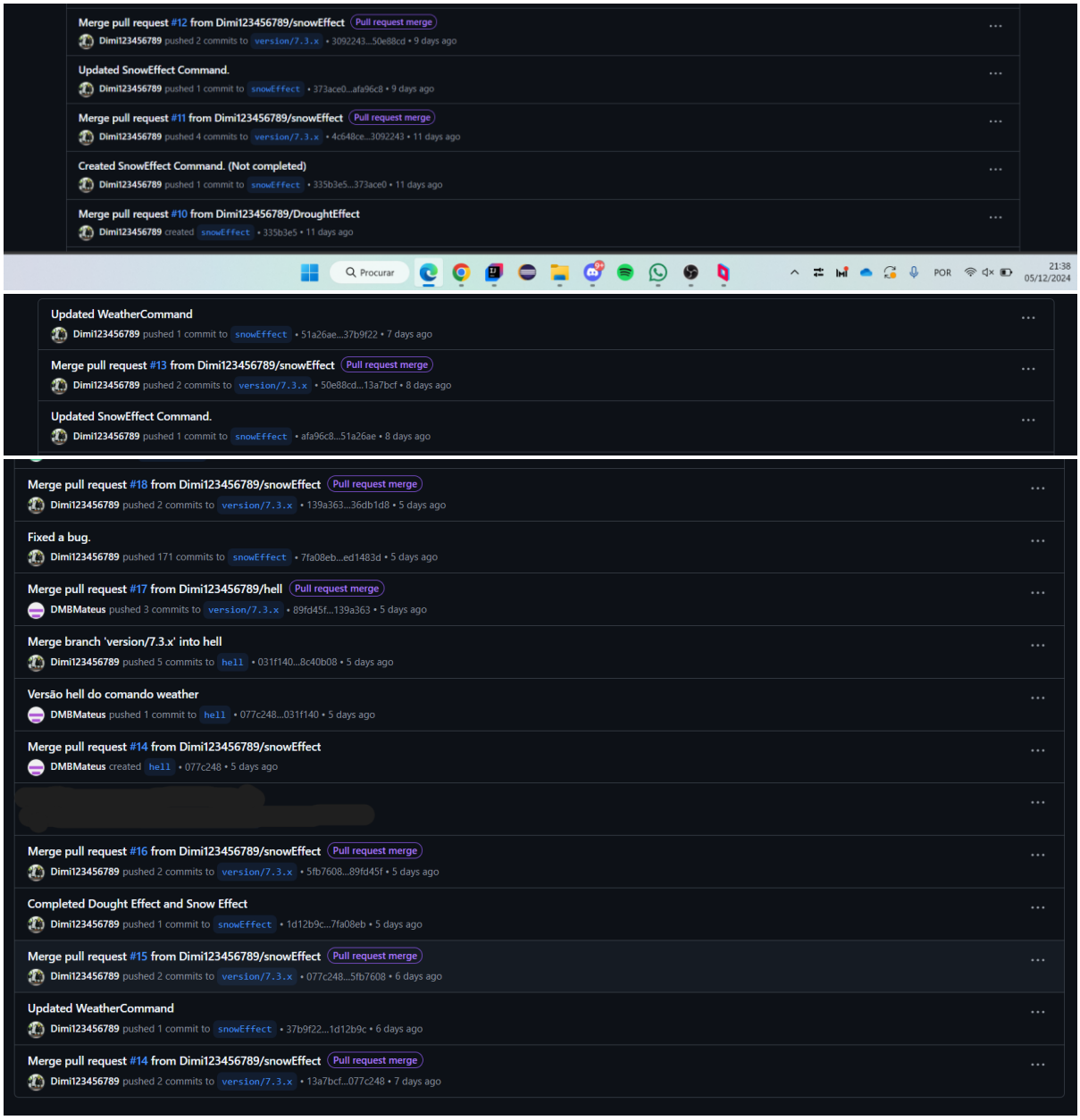
Pre-condition: Logged in
Test case id: Snow Effect ok
Steps: 1. Download the schematic : “finalProject.schem” (available on our gitHub repository) (if already downloaded ignore this step)
2. Put the “finalProject.schem” file in the schematics folder (the path is: worldedit-fabric\run\config\worldedit\schematics)
3. In minecraft type: “//schematic load finalProject schem”
4. In minecraft type: “//paste”
5. With your Wooden Axe left click the leave block situated on the top left corner of the loaded building
6. With your Wooden Axe right click the yellow concrete block situated on the bottom right corner of the loaded building
7. In minecraft type: //weather rain
Expected results: 1. You should have a file on your computer named “finalProject.schem”
2. Is expected to have no errors when adding the file to the folder
3. In chat should appear a message saying: “finalProject loaded. Paste it with //paste”
4. The building is generated near the player
5. In chat should appear a message saying: “First position set to (x, y ,z) (n)”
6. In chat should appear a message saying: “Second position set to (x, y ,z) (n)”
7. The selected area changes and is printed a message saying “Transformed the blocks to simulate a rainy environment!”

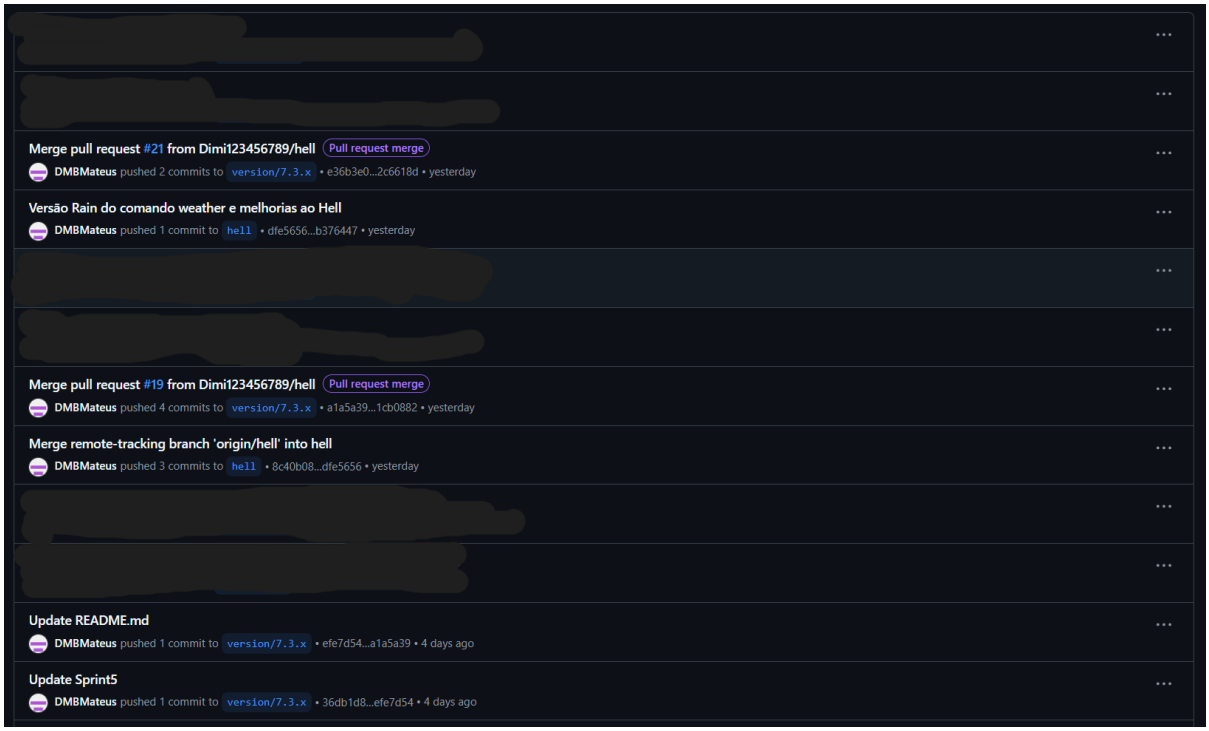
Tour report:

To implement this command we created a method in the final lines of the RegionCommands Class(1) called weather() (line 624). Here we use a switch case to call for each different subcommand of the weather command. Each one calls for a different method in the Class EditSession(2). The four methods needed can be found after line 2979. In each one of this methods, it’s created an object corresponding to the subcommand in question. In total, 4 Classes that we implemented are created here: DroughtEffect(3), SnowEffect(4), HellEffect(5) and RainEffect(6). Finally, to add the messages that appear in chat, we moddified strings.json(7) adding the changes in the last lines of the code.

<u>All Paths to the mentioned Classes:</u>
(1) worldedit-core/src/main/java/com/sk89q/worldedit/command/RegionCommands.java
(2) worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java
(3) worldedit-core/src/main/java/com/sk89q/worldedit/function/block/DroughtEffect.java
(4) worldedit-core/src/main/java/com/sk89q/worldedit/function/block/SnowEffect.java
(5) worldedit-core/src/main/java/com/sk89q/worldedit/function/block/HellEffect.java
(6) worldedit-core/src/main/java/com/sk89q/worldedit/function/block/RainEffect.java
(7) worldedit-core/src/main/resources/lang/strings.json

Dimitrios was responsible by the Snow and Drought subcommands, while Diogo was responsible for the Hell and Rain subcommands. As such, here are the commits made to prove the claimed changes as well as the division of the work (only commits made by either Dimitrios or Diogo will be visible) :





(NOTE: all code made by Diogo was reviewed by Dimitrios and all code made by Dimitrios was reviewed by Diogo)

//lake <radii> <type>

Name: Create lake

ID:2

Brief Description: Generate a lake on the position the player is currently at between four different types of lake (water, lava, oasis, pond, infinite)

Primary Actors: Player

Secondary Actors: None

Preconditions:

1. The player is on the ground
2. The player is on flat terrain
3. The player is a valid player
4. The editSession is the current session we are working in

Main Flow:

1. The use case starts when the player uses the command //lake <radii> <type>
2. The system validates the player position, radii and lake type, if valid the type of lake chosen by the player will be generated on the player’s position, if not an error message will appear in the player's chat
3. The system counts and logs the number of affected blocks

4. A success message is displayed to the player referencing which type of lake was created

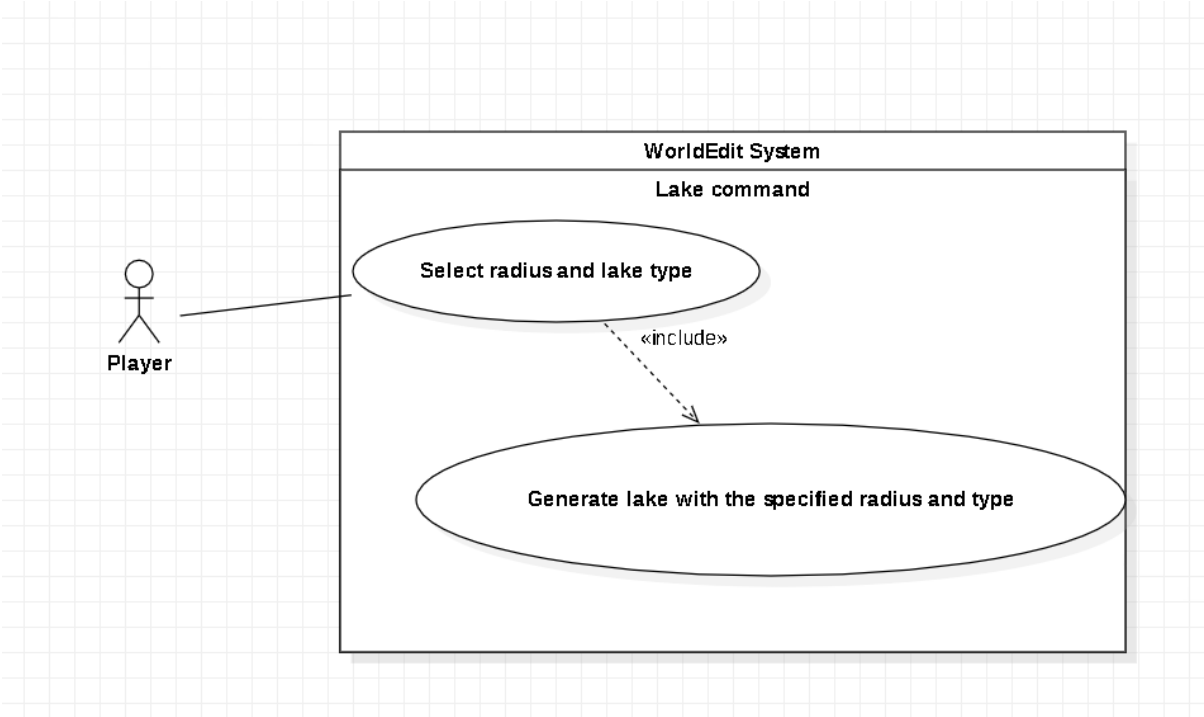
Post Conditions:

1. A message confirming the operation or indicating an error is displayed to the player
2. A lake is generated on the player's position

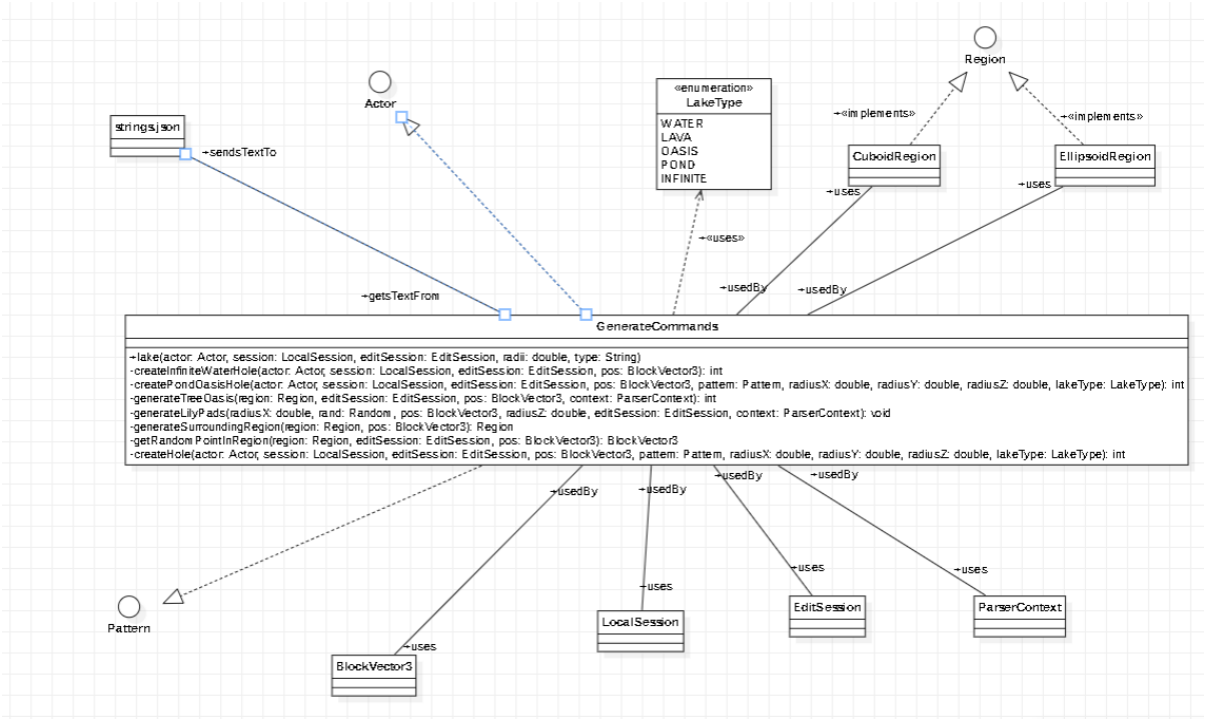
Alternative flows:

1. Invalid Player Position: If the player is not currently on the ground, the system displays an error message: ("Command can only be used while standing on the ground")
2. Invalid Radii: If the player inputs a radius inferior to 5, the system displays an error message: ("Radius too small, radius should be greater than or equal to 5")
3. Invalid Lake Type: If the player inputs a lake type that is not one of the supported lake types, the system displays an error message: ("Invalid lake type, try: [water | lava | pond | oasis | infinite]")

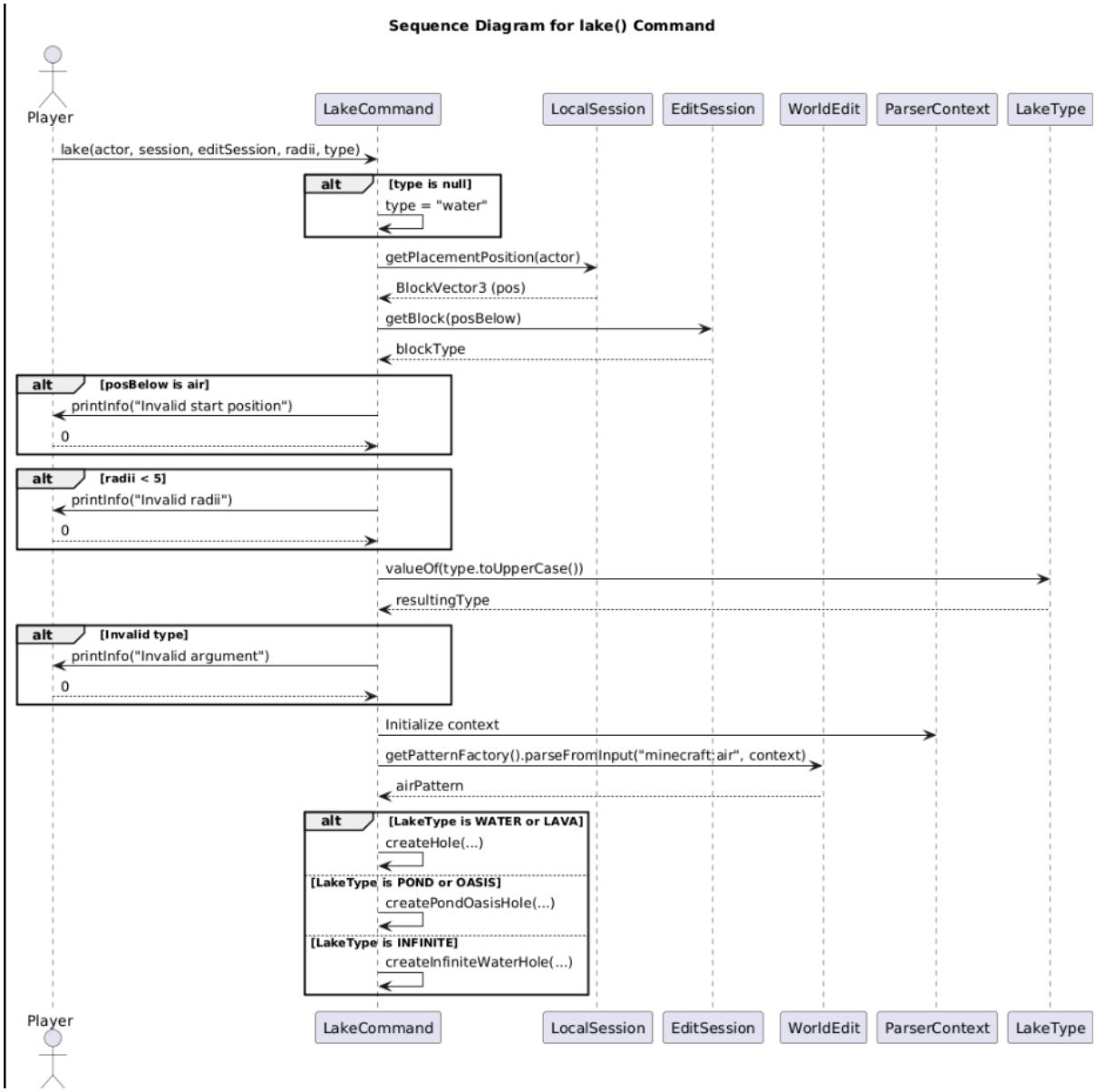
Use case diagram:



Class Diagram:



Sequence Diagram:



Scenario Based Testing:

- **Water:**

Pre-condition: Logged in
Test case id: Water Lake ok
Steps: 1. Place the character where you want the lake to be generated;
2. In chat type //lake <radii> water
Expected results:
1. Is expected to have no errors when creating the lake;
2. In chat should appear a message referencing which type of lake was created

- **Lava:**

Pre-condition: Logged in

Test case id: Lava Lake ok
Steps: 1. Place the character where you want the lake to be generated;
2. In chat type //lake <radii> lava
Expected results:
1. Is expected to have no errors when creating the lake;
2. In chat should appear a message referencing which type of lake was created

- **Pond:**

Pre-condition: Logged in
Test case id: Pond Lake ok
Steps: 1. Place the character where you want the lake to be generated;
2. In chat type //lake <radii> pond
Expected results:
1. Is expected to have no errors when creating the lake;
2. In chat should appear a message referencing which type of lake was created

- **Oasis:**

Pre-condition: Logged in
Test case id: Oasis Lake ok
Steps: 1. Place the character where you want the lake to be generated;
2. In chat type //lake <radii> oasis
Expected results:
1. Is expected to have no errors when creating the lake;
2. In chat should appear a message referencing which type of lake was created

- **Infinite:**

Pre-condition: Logged in
Test case id: Infinite Lake ok
Steps: 1. Place the character where you want the lake to be generated;
2. In chat type //lake <radii> Infinite
Expected results:
1. Is expected to have no errors when creating the lake;
2. In chat should appear a message referencing which type of lake was created

Tour Report:

To implement this command we created a method in the final lines of the GenerationCommands Class(1) called lake() (line 497). Here we use a switch case to call for each different subcommand of the lake command, of the 5 types, 2 pairs call the same function(lava,water call createHole() (line 694) and pond, oasis call createPondOasisHole() (line 575)) the other type (infinite) calls the method createInfiniteWaterHole() (line 557). We also made some auxiliary methods and an enum called generateTreeOasis() (line 654), generateLilyPads() (line654), generateSurroundingRegion() (line 667),

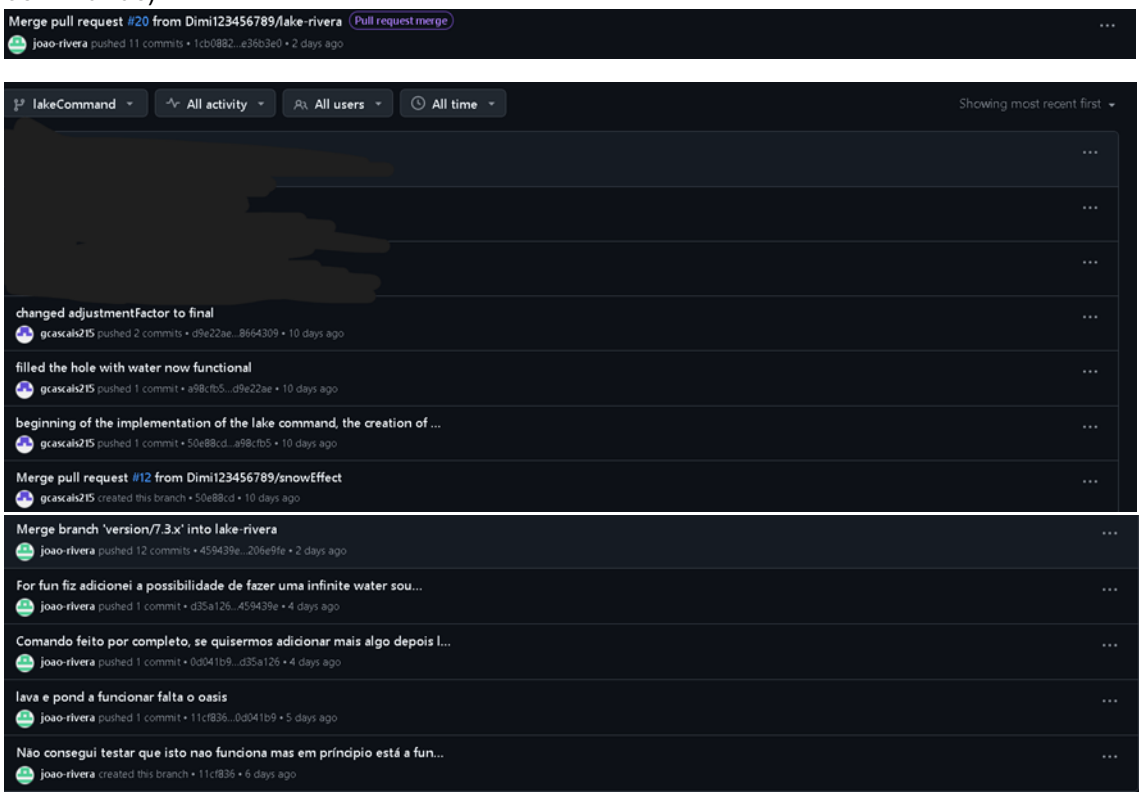
getRandomPointInRegion() (line 676). The messages that appear in chat were added to the strings.json(2) file.

All paths to the mentioned Classes:

(1) worldedit-core/src/main/java/com/sk89q/worldedit/command/GenerationCommands.java

(2) worldedit-core/src/main/resources/lang/strings.json

Gonçalo was responsible for the base of the command and the water type, Rivera was responsible for the lava, pond, oasis, infinite types. The commits were as follows(the blurred out parts are either commits unrelated to the code or made by other members for other commands):



Link for the Demo Video:
https://www.youtube.com/watch?v=SiuiR4af_FY

Conclusions

The project provided a valuable opportunity to apply theoretical knowledge of software engineering principles to a real-world scenario. By developing and integrating advanced features into the WorldEdit plugin, the team demonstrated a thorough understanding of object-oriented programming concepts, design

patterns, and code quality assessment. Each milestone represented a critical step towards achieving a modular, efficient, and user-friendly tool.

Analysing the implementation of non-trivial design patterns, such as Singleton, Factory, and Adapter, allowed us to understand how to improve the maintainability and scalability of the system. The codebase metrics and code smells analysis highlighted potential areas for optimization, guiding efforts to refactor complex or tightly coupled components. This approach ensured that the final code adhered to best practices, enhancing both readability and extensibility.

Furthermore, the iterative development process supported by Scrum methodologies enabled effective team collaboration and task prioritization. The use of burndown charts and scrum boards facilitated transparent tracking of progress and identification of bottlenecks, ensuring timely delivery of project goals.

Overall, the project successfully combined technical skills, teamwork, and analytical approaches to deliver a functional and robust solution. The lessons learned will undoubtedly contribute to the team’s future endeavors in software development, fostering a culture of continuous improvement and adherence to high-quality standards.