**KSU Minecraft ESports Speedrun Plugin**

# Design Document

# (Christopher Kirkwood, Weeldy Benjamin)

Date: 09/22/2024

Kennesaw State University College of Computing and Sof

Course: SWE Capstone

Professor: Yan Huang

**TABLE OF CONTENTS**

1. INTRODUCTION

1.1 Purpose

1.2 Scope

1.3 Overview

1.4 Reference Material

1.5 Definitions and Acronyms

2. SYSTEM OVERVIEW

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

3.1.1 UML Component Diagram

3.2 Technical Design

3.3 Design Rationale

4. DETAILED DESIGN

4.1 UML Class Diagram

4.2 UML Class Diagram Description

5. DATABASE (DATA) DESIGN

6. HUMAN INTERFACE DESIGN

6.1 UI design

6.2 UX design

7. REQUIREMENTS MATRIX

8. APPENDICES

### INTRODUCTION

## Purpose

This software design document describes the architecture and system design of the Kennesaw State University Esports Minecraft Speedrun Plugin, “KSU-Minecraft-Speedrun”. This includes an overview of the functionality, context, design, and requirements of this project.

​​This document is intended for use by the KSU Esports sponsor, KSU College of Computing and Software Engineering faculty, as well as the members of SWE Capstone Team 15-1: Adam Greene, Chris Kirkwood, Rachel Amponsah, Steven Kelsey, and Weeldy Benjamin. These stakeholders have not expressed any specific security or privacy considerations associated with the use or disclosure of this document; however, general principles of data protection and proper document handling should be followed.

## Scope

The objective of this project is to create a fun speedrun Minecraft game mode for KSU students that utilizes the current version of Minecraft as well as the Paper API and promotes the server’s overall usability, reliability, and scalability for both players and administrators. The software itself must be developed with optimized efficiency and performance – handling all logic off the main server thread, when possible – as well as maintainability and compatibility, supporting future versions of Minecraft and the Paper API with as minimal disruption as possible.

The product being delivered for this project is a Speedrun plugin, where players will race to complete an objective or a set of objectives within a specified time-period in Minecraft. The plugin will be highly customizable, allowing for different attributes, such as world borders, objectives, and game rules to be modified at the will of the administrators.

## Overview

This document contains a general description of the functionality, context, and design of this project, as well as specific details regarding the architectural and technical design of the project and the rationale behind design selection. The document will outline the conceptual structure of each component, the relationship between them, and how it contributes to the functionality of the system as a whole. Furthermore, details regarding UI design and how data storage will be handled will all be outlined. Functional Requirements will be demonstrated as well as their relationship to components in the system.

***Please view the next page..***

## Reference Material

Some material from this document was sourced from the preceding deliverable from Team 15-1, the Project Plan (Form-02), titled “TEAM-15-1-SCP-FORM-02”. No other reference material was used.

## Definitions and Acronyms

* **IDE (Integrated Development Environment)**: A software application, like IntelliJ IDEA, is used by developers to write, test, and debug code.
* **Paper API**: A set of programming interfaces, which is used to extend or modify Minecraft gameplay via plugins.
* **UI:** User Interface – An interface that provides enhanced usability.
* **GUI:** Graphical User Interface - In this case, the term is interchangeable with UI.
* **UML:** Unified Modeling Language - A visual language that assists in visualizing systems.

***Please view the next page..***

### SYSTEM OVERVIEW

The “KSU-MC-Speedrun” plugin is designed to create a customizable speedrun gamemode within Minecraft 1.21+ using the Paper API. This plugin seeks to deliver an engaging and fun experience for players while easing the stress of server administrators during speedrun events. The main functionality of the plugin is to streamline the process of running speedrun events on the KSU Esports Minecraft server—optimizing scalability and usability for operators and improving the reliability of the system for both operators and players alike.

There are four main objective types that can be assigned to a speedrun: kill, mine, obtain, and enter. Event listeners listen for these events, and if they match the objective's requirements, the system marks the objective as complete. In addition, there are many other attributes of the speedrun that are fully customizable within the system. This includes time limits, game rules, world borders, and player spawn radii. The system also supports advanced configuration through the config.yml file, where settings like message customization and structure proximity can be modified.

The plugin design is highly maintainable. Objectives are organized around an abstract Objective class, with specific objective types (such as EnterObjective, KillObjective, MineObjective, and ObtainObjective) extending from this base. Custom objects like Portal and SRStructure help fill gaps in the Paper API, enabling the system to track custom objectives. Objectives are managed through an ObjectiveManager, and teams of players can be organized and tracked via a TeamManager.

All components of the system are brought together in the Speedrun class, which serves as the central controller for managing the game mode. Commands issued by server administrators allow for full control over the speedrun, including objective management and game settings. Assuming development of the project continues in a timely manner, the final deliverable will also include a UI for administrators and players as an alternative to using commands.

### SYSTEM ARCHITECTURE

## Architectural Design

The architecture of the “KSU-MC-Speedrun” plugin is designed to promote maintainability, supporting future versions of Minecraft and the Paper API. The system is broken down into multiple components, each responsible for specific tasks, that collaborate with each other to achieve complete functionality of the speedrun gamemode. Below is a high-level overview of the main components included in the plugin.

1. Main System (Main Class)
2. Configuration Subsystem (Config Class)
3. Speedrun Subsystem (Speedrun Class)
4. Team Manager Subsystem (Teams Package)
5. Objective Manager Subsystem (Objectives Package)
6. Event Listener Subsystem (Events Package)
7. Command Subsystem (Commands Package)
8. Map Verification Subsystem (MapVerification Class)
9. Other Custom Objects (Structures Package)

The main system works as the entrypoint and controller of the plugin lifecycle. When the plugin is first enabled, the *onEnable()* method in the main class is called. Within this method, all of the other high-level subsystems are initialized. This includes the configuration subsystem, which has direct access to load/generate, read, and write to the *Config.yml* file, as well as the speedrun subsystem, which is the central subsystem that manages the speedrun’s state, objectives, teams, and settings. Also included within the *onEnable()* method in the main system are calls to register all event listeners as well as the command subsystem, which enables the command */speedrun* from within the server.

The Speedrun subsystem, upon initialization, subsequently creates all private variables pertaining to the speedrun’s state, objectives, teams, and settings, which includes initialization of both the objective manager subsystem as well as the team manager subsystem and map verification subsystem. The objective manager subsystem contains a list of abstract *Objective* objects that can be added to or removed from via the main speedrun instance. Likewise, the team manager subsystem contains a list of *Team* objects that can be added to or removed from via the main speedrun instance. Each individual *Team* object contains a list of players that can be adjusted. Use of modular subsystems such as the objective manager or team manager ensure that new objectives or teams can be added without major changes to the software design, thus promoting maintainability.

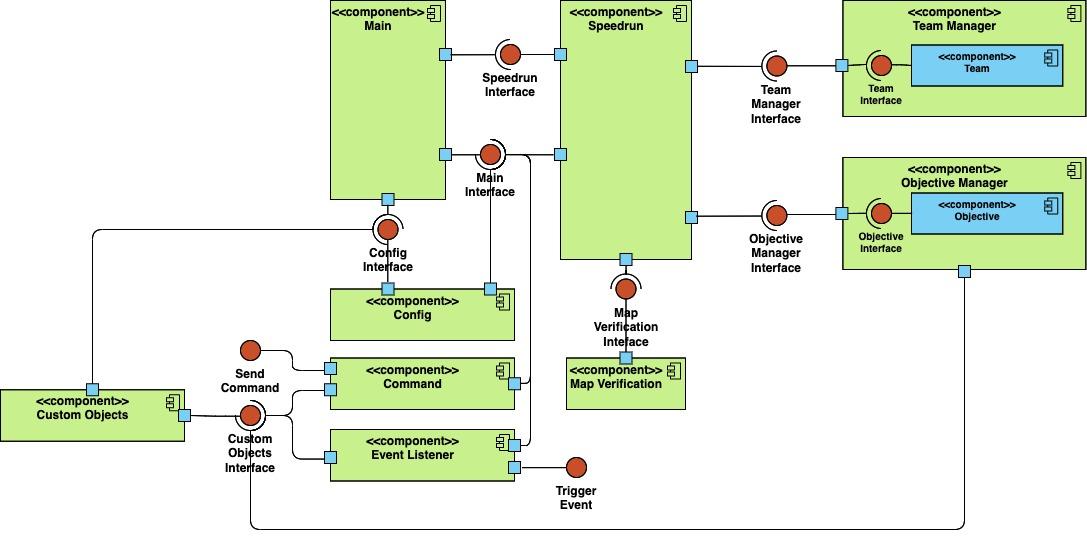
The map verification subsystem is implemented by the speedrun subsystem to verify that world seeds are compatible with objectives within the specified world border. Thus, the speedrun subsystem also works as a liaison between the map verification subsystem and the objective subsystem.

The event listener subsystem works closely with the speedrun subsystem (particularly the objective subsystem and team subsystem within the speedrun subsystem) to detect if a relevant event has occurred (*kill*, *mine*, *obtain*, or *enter*), and uses a variety of conditional checks to determine whether the event satisfies an objective requirement. If so, an event is marked as complete, with the “completer” being the team of which the player that completed the objective belongs to.

The command subsystem works as an interface between in-game (or console) server administrators and the speedrun subsystem. This subsystem allows administrators to access all of the functionality of the speedrun system through in-game commands.

Finally, there are some other custom objects that are implemented to allow for communication between the *EnterObjective* (extension of abstract *Objective*) and the command subsystem with the Paper API. *SRStructure* is an example of a custom object which implements a functionality to get the Paper API *Structure* object from a string value, which is not supported by default. These objects also work with the configuration subsystem to promote maintainability for future updates that add new dimensions or structures to the game.

**3.1.1 UML Component Diagram**



The diagram above provides a high-level overview of the system's design. The Speedrun and Config components are initialized by the Main plugin, which passes its instance into these components for use throughout the system. The Main plugin instance is also passed to the Command and Event Listener components, allowing access to the Config and Speedrun instances. The ObjectiveManager and TeamManager are both initialized through the Speedrun component and can be interfaced with to manage objectives and teams during a speedrun. The Speedrun component also interacts with the Map Verification component to verify map compatibility (e.g., world seeds and map boundaries). Lastly, the Custom Objects component provides an interface for the Command, Event Listener, and Objective Manager components to bridge functionality gaps where the Paper API does not offer sufficient support, such as with custom structures and portals.

## Technical design

The “KSU-MC-Speedrun” plugin utilizes several different technologies to create a customizable speedrun gamemode for Minecraft 1.21+. At the core, the plugin is built using the Paper API, which is an interface that allows for server-side Minecraft modifications.

Aside from the libraries provided by the Paper API, the plugin utilizes various Java utilities such as **java.util.UUID**, **java.util.HashMap**, **java.util.List**, **java.util.Random**, as well as some i/o utilities, including **java.io.File** and **java.io.PrintWriter** for configuration management. The software is developed using the “Minecraft Development” plugin in IntelliJ IDEA; builds and dependencies are managed with Gradle build tools.

Async methods are used within the plugin wherever possible: in the Paper API, async threads are not recommended for any operations that interact directly with players or world data. This is taken into careful consideration when developing this plugin - ensuring that any time consuming or resource intensive operations are handled off the main thread, but that the main thread is always used for world/player interactions.

## Design Rationale

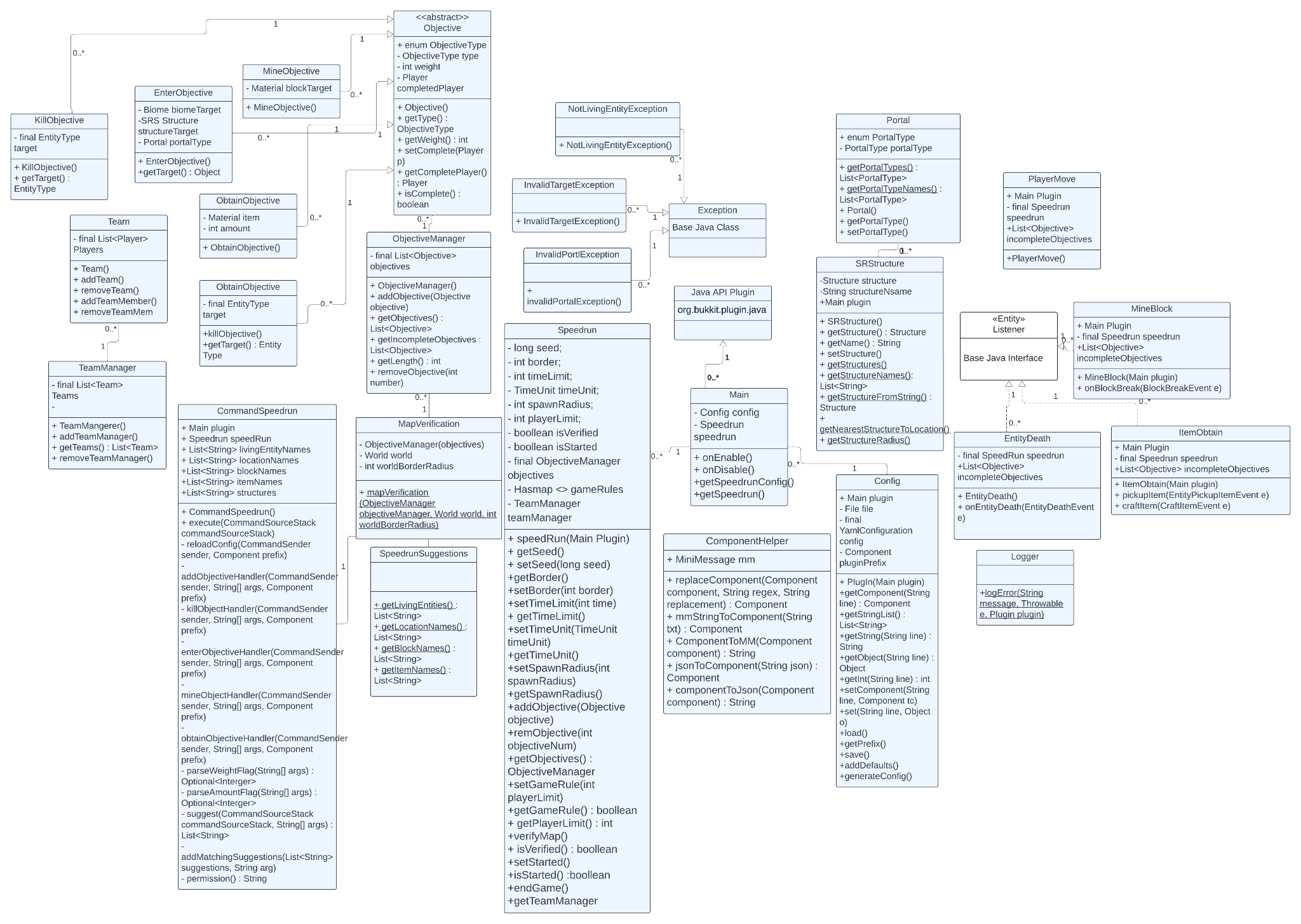
This architecture chosen follows a modular, component-based design that prioritizes maintainability and scalability. The architecture utilizes various subcomponents such as the Speedrun, Objective Manager, Team Manager, and Config subsystems, of which they each have very clearly designed roles. The rationale behind this choice was to ensure that each component can operate independently and be modified/fully replaced without having a greater negative impact to the entire system. This allows for increased support for future updates to the system.

The Paper API, which is built using the Java programming language, was a requirement set by the sponsor of this project. Thus, Java is the programming used for this project based on that requirement. The IDE of choice, IntelliJ IDEA, was selected due to its support for Minecraft development. To support the architectural design of the plugin, principles of Object-Oriented design are to be implemented in the software, including inheritance, abstraction, and polymorphism.

***Please view the next page..***

### DETAILED DESIGN

**4.1 UML Class Diagram**



**4.2 UML Class Diagram Description**

* The main component provides instances of the Speedrun and config components and has as main function to act as a central hub for the plugin API. Config Manager manages the plugin config which handles loading, saving, and updating of settings.
* The Speedrun/Speedrun Manager acts as a controller for the speedrun game mode and integrates with ObjectiveManager to manage objectives for the speedrun.
* The abstract class Objective serves as the base for all types of objectives which are: KillObjective, EnterObjective, MineObjective, ObtainObjective. The class also connects directly to the ObjectiveManager class which manages the collection of objectives.
* Event listeners are used and help to create things like EntityDeath, PlayerMove, MineBlock, and ItemObtain events which are all events tracked and monitored by the listeners. Additional event listeners may include inventory events and player chat events.
* The Command System is used to help server admins to control and configure the game. The Command System utilizes commands such as addObjective, time, setSeed, timeLimit, spawnRadius, playerLimit, start, forceStart commands
* Team System is created via the teams class and is controlled by the TeamManager class. This is primarily used to allow players to create teams.
* Utility Classes are classes that are used to format messages and provide suggestions for speedrun settings.
* SRStructures and Portal classes manage structure types in game and allow the user to locate structures in game. These classes also support objectives to be added as an attribute to the structure.
* There are also three exception classes currently used to handle errors related to objectives. The exceptions are primarily used during objective creation to ensure valid enforcement of rules ect.

### DATA (DATABASE) DESIGN

Given how little information is stored during the runtime of the “KSU-MC-Speedrun” plugin and the nature of this project being a modification to the minecraft server software, there is no need for the addition of a traditional database.

The plugin will reference a single YamlConfiguration file that may be modified outside of the program or during using the command UI through the already existing Minecraft chat system or through a specialized speedrun admin UI described in the next section.

This addon will reference specific users' security information given by the original KSU Minecraft server, but this information will not be modified or re-recorded in this program.

### HUMAN INTERFACE DESIGN

## UI design

The UI design of the “KSU-MC-Speedrun” plugin focuses on delivering intuitive interfaces for players and administrators, ensuring that the plugin is easy to use while the user interacts with the speed run system. Below are the main UI elements:

1. **Player Interface**
   1. **Chat Interface:** Players will use in-game chat to communicate with their team. Messages will also be color-coded based on whether they are team or global messages.
   2. **Team Chat:** Private messages between team members, accessible via /teamchat.
   3. **Global Chat:** Public messages to all players, controlled by the administrator.
   4. **Status Bar:** Displays the player's current status, alongside speed run-specific stats such as objective progress and completion percentage.
   5. **Team Selection**: Players can select their teams through an inventory UI prior to the start of the speedrun.
2. **Administrator Interface**
   1. **Command Panel:** Administrators can control the game mode using in-game commands. The administrator's GUI may provide a simplified control panel to manage settings like objectives, time limits, and teams.
   2. **Objective Management UI:** The graphical Minecraft interface that allows admins to add, remove, or modify objectives anytime during the game.
   3. **Settings GUI:** This will allow admins to change game rules, world borders, and spawn points at the beginning of the game. Settings such as team size, allowed objectives, and time limits may be adjusted before and during the game.
   4. **Real-Time Feedback:** Admins will receive alerts when teams complete objectives or when certain game conditions are triggered. An example of a condition would be time running out.

## UX design

The UX design of the KSU-MC-Speed run plugin focuses on optimizing player engagement and administrator ease of use. The experience should be fun and efficient for everyone involved.

1. **Player Experience**
   1. **Smooth Onboarding:** Players will receive clear instructions upon joining the server about how this game mode works. An introduction command, like /speedrun help, will explain objectives and controls.
   2. **Real-time Feedback:** Players will receive real-time notifications and visual feedback during the game.
   3. **Team Collaboration:** The UX is centered around collaboration, with the shared inventory system, and the chat system allowing for teamwork and quick communication.
   4. **Minimal Distractions:** The interface will be kept minimal and non-intrusive. Only essential information will be displayed to avoid cluttering the player's screen.
2. **Administrator Experience**
   1. **Easy Setup and Management:** Administrators will have an intuitive setup process for configuring game rules and settings. The setup will be fast and simple.
   2. **Control Over Game Progress:** Administrators will have the ability to pause, restart, or modify objectives in real time. They will also receive real-time alerts on game events.
   3. **Customization:** Admins will be able to customize aspects of the speed run event, from objective types to world settings and player limitations, with minimal hassle.
   4. **Comprehensive Logs**: Admins will have access to detailed logs of player progress, objective completions, and errors. This helps in monitoring and debugging.

### REQUIREMENTS MATRIX

| Requirement ID | Name | Description | Priority | Component |
| --- | --- | --- | --- | --- |
| FR1 | OBJECTIVETYPES | The plugin shall support four different objective types: kill an entity, mine a block, obtain an item, or enter a location. | HIGH | Objective Manager |
| FR2 | ENTEROBJECTIVE | Enter objectives shall support three different location types: Biome, Portal, or Structure. | HIGH | Objective Manager |
| FR3 | KILLOBJECTIVE | Kill objectives shall support all living entity types in the Paper API. | HIGH | Objective Manager |
| FR4 | OBTAINOBJECTIVE | Obtain objectives shall support all materials in the Paper API. | HIGH | Objective Manager |
| FR5 | MINEOBJECTIVE | Mine objectives shall support all materials in the Paper API that pass the isBlock() condition. | HIGH | Objective Manager |
| FR6 | EVENTLISTENER | The plugin shall implement event listeners to listen for all objective events and communicate with the objective manager if an event is completed. | HIGH | Event Listener |
| FR7 | KILLEVENTVERIFY | The plugin shall be capable of attributing indirect kills to players where default Minecraft does not (i.e., fall damage kills, bed explosions, and end crystal explosions) | HIGH | Event Listener |
| FR8 | OBJECTIVEAMOUNT | The plugin shall not have a set maximum amount of objectives that can be assigned. | HIGH | Objective Manager |
| FR9 | OBJECTIVEMANAGEMENT | The plugin shall allow administrators to dynamically add or remove objectives during the speedrun event. | HIGH | Objective Manager |
| FR10 | FUNCTIONS | The plugin shall implement support for all speedrun functions (see appendix a) and team functions (see appendix b). | HIGH | Speedrun |
| FR11 | COMMANDS | The plugin shall implement chat commands for all speedrun functions (see appendix a) and team functions (see appendix b). | HIGH | Command |
| FR12 | PERMISSIONS | The plugin shall implement permission nodes to allow players and administrators to access player commands and administrators to access administrator commands (see appendix a & b). | HIGH | Command |
| FR13 | SUGGESTIONS | The plugin shall implement command suggestions for all subcommands included within the main speedrun command. | HIGH | Command |
| FR14 | SPAWNTEAMS | The plugin shall spawn players on the same team at the same location. | HIGH | Speedrun |
| FR15 | SPAWNSPACE | The plugin shall spawn teams equidistantly from each other around the specified radius, which will depend on how many teams are in use. | HIGH | Speedrun |
| FR16 | TEAMGENERAL | The plugin shall implement 16 preset teams, one for each wool color in Minecraft. | HIGH | Team Manager |
| FR17 | TEAMAMOUNT | The plugin shall calculate how many teams are available by dividing the player count by the set team size (see appendix c). | HIGH | Team Manager |
| FR18 | GAMERULES | The plugin shall allow administrators to define custom game rules for the speedrun event. | HIGH | Speedrun |
| FR19 | TIMECONTROL | The plugin shall display the remaining time on screen via an objective (net.minecraft objective) on the right side of the screen, or as a boss bar. | HIGH | Speedrun, Command |
| FR20 | LEADERBOARD | The plugin shall implement a point system to track the progress of teams based on their completed objectives. | HIGH | Speedrun, Objective Manager |
| FR21 | TEAMUI | The plugin shall implement a user interface as an alternative to using team subcommands using an inventory screen and a specific block type for every team (see appendix b). | MEDIUM | Event Listener |
| FR22 | SPEEDRUNUI | The plugin shall implement an administrative user interface as an alternative to using speedrun subcommands using an inventory screen and a specific block type for every command (see appendix a). | LOW | Event Listener |
| FR23 | WORLDVERIFY | The plugin shall verify that all objectives set can be completed by in the provided world seed within the specified world border. | HIGH | Map Verification |
| FR24 | WEIGHT | The plugin shall implement functionality to set the weight of a specific objective above the default value of 1. | HIGH | Objective Manager |
| FR25 | AMOUNT | The plugin shall implement functionality to set the amount of items required to satisfy an OBTAIN objective. | HIGH | Objective Manager |
| FR26 | ANNOUNCEMENT | The plugin shall broadcast automated in-game announcements for significant events, such as objective completion, new objectives, and time updates | MEDIUM | Event Listener |
| FR27 | TEAMCHAT | The plugin shall allow for players to switch between teamchat and globalchat using the command /teamchat. | LOW | Command |
| FR28 | CHATDISPLAY | The plugin shall only display teamchat messages to members of the message sender’s team. | LOW | Event Listener |
| FR29 | RESETGAME | The plugin shall allow administrative users to reset all progress, objectives, and timers and return players to the spawn location. | HIGH | Speedrun |
| FR30 | MINIMESSAGE | The plugin shall allow customization of plugin messages in the config.yml using MiniMessage format. | HIGH | Config |
| FR31 | STRUCTURECONFIG | The plugin shall allow customization of structure detection radii through the config.yml. | HIGH | Config |
| FR32 | PORTALCONFIG | The plugin shall allow support for configuration of future portal types through the config.yml to support future Minecraft versions in the event of a new dimension being added. | LOW | Config |
| FR33 | REGISTEREVENTS | The plugin shall register all event listeners upon startup. | HIGH | Main |
| FR34 | REGISTERCMDS | The plugin shall register the /speedrun command upon startup. | HIGH | Main |
| FR35 | REGISTERSPDRUN | The plugin shall create an instance of Speedrun upon startup that can be returned through the main instance. | HIGH | Main |
| FR36 | GENCONFIG | The plugin shall generate a config.yml file with default values if one does not exist upon startup. | HIGH | Main |
| FR37 | LOADCONFIG | The plugin shall load the config.yml file file upon startup if it exists. | HIGH | Main |
| FR38 | REGISTERCONFIG | The plugin shall create an instance of the loaded config file upon startup that can be returned through the main instance. | HIGH | Main |

### APPENDICES

1. Speedrun Functions
   1. Add Objective - admin
   2. Remove Objective - admin
   3. Get Objectives - player
   4. Set Time Limit - admin
   5. Get Time Remaining - player
   6. Set World Border - admin
   7. Set Seed - admin
   8. Set Player Limit - admin
   9. Set Spawn Radius - admin
   10. Set Team Size - admin
   11. Set Required Points for Completion - admin
   12. Add Gamerule - admin
   13. Remove Gamerule - admin
   14. Verify Map - admin
   15. Start Speedrun - admin
   16. Stop Speedrun - admin
   17. Get Leaderboard
   18. Help - player
2. Team Functions
   1. Join Team - player
   2. Change Team - player
   3. Add Team - console (no admin command support)
   4. Remove Team - console (no admin command support)
   5. List Teams - player
3. Team Calculations
   1. The amount of teams available shall be up to 16, but will change dynamically depending on the team size set by administrators and the player count of the server:
      1. # of teams available = ⌈player count / team size limit⌉
      2. e.g.: # of teams available = ⌈50 / 5⌉ = 10
      3. e.g.: # of teams available = ⌈51 / 5⌉ = 11
   2. If the # of teams available exceeds 16, the plugin shall notify an administrator that the team size limit must be increased:
      1. # of teams available = ⌈100 / 6⌉ = 17;
      2. 17 > 16, team size limit is invalid
      3. Administrators will be notified that the team size limit must be increased to 7 before the game begins.
   3. The amount of teams available for players to join will be based on the number calculated, from 1 to 16.