

03-T2 Minecraft

Final Report

CS 4850 – Section 01 – Spring 2026

May 4, 2026

<https://github.com/minecraft-turf-wars/minecraft-turf-wars>

<https://minecraft-turf-wars.netlify.app>

Name	Role
Matthew Pflieger (Team Leader)	Developer
Chase Holt	Documentation
Andrew Burgos	Developer
Hezekiah Holloman	Documentation
John McDowell	Developer
Sharon Perry	Project Owner or Advisor

STATS AND STATUS	
Lines of Code (LOC)	1,243
Components/Tools	Java, PaperMC, Gradle
Hours Estimate	275
Hours Actual	270
Status	100%

Table of Contents

Introduction / Abstract.....	4
Project Plan.....	4
Project Overview.....	4
Project Website.....	4
Deliverables.....	4
Group Meeting Schedule Date/Time.....	4
Collaboration and Communication Plan.....	4
Project Schedule and Task Planning.....	5
Other Plans.....	5
Version Control Plan.....	6
Software Requirements Specification (SRS).....	6
Overview.....	6
Project Goals.....	6
Definitions and Acronyms.....	6
PaperMC: a Spigot server implementation.....	6
Assumptions.....	6
Design Constraints.....	7
Environment.....	7
User Characteristics.....	7
Functional Requirements.....	7
Game State & Lobby Management.....	7
Turf Game Mode Mechanics.....	7
Combat Mechanics.....	7
Team Balancing.....	8
Non-Functional Requirements.....	8
Security.....	8
Capacity.....	8
Usability.....	8
Other.....	8
Software Design Document (SDD).....	9
Design Considerations.....	9
Architectural Strategies.....	9
Programming Language.....	9
PaperMC.....	9
Future Plans.....	9
System Architecture.....	9
Detailed System Design.....	9
Classification.....	9
Constraints.....	10
Resources.....	10

Glossary.....	10
Development Document.....	11
Implementation Detail Narrative.....	11
System Integration & Workflow Overview.....	11
Arena Manager.....	11
Game Manager.....	11
Database and External Connections.....	11
Project Setup and Environment Configuration.....	12
Prerequisites.....	12
Installation Steps.....	12
Verification.....	12
Game Design Document.....	13
Game Overview.....	13
Gameplay.....	13
Objectives.....	13
Game Progression and Play Flow.....	13
Mission Challenge or Puzzle Structure.....	13
Game World.....	13
Characters and Opponents.....	14
Levels.....	14
Deployment.....	14
Development.....	14
Test Plan and Report.....	15
Overview.....	15
Purpose.....	15
Scope.....	15
Scope of Testing.....	15
In Scope.....	15
Out of Scope.....	15
Test Cases.....	16
Test Procedures.....	17
Test Environment.....	17
Software Test Report (STR).....	18
Version Control.....	18
Conclusion or Summary.....	19
Appendix –Training Verification.....	19

Introduction / Abstract

Turf Wars is a fast paced, team based, player versus player experience that combines elements of a traditional “tug-of-war” game with Minecraft’s building and combat mechanics. Gold Team and Black Team face off in an arena where the ground they stand on represents their score in the game. The objective is simple: expand your team’s territory by eliminating enemy opponents.

Kennesaw State University Esports needed a fresh and engaging Minecraft gamemode for campus events. This game will satisfy all of these requirements helping drive more players to the KSU Minecraft server. This gamemode was purposefully designed with simplicity in mind so experienced players and new players will be able to enjoy the game together.

Project Plan

Project Overview

For this project we are creating a Minecraft game mode. This will be used in Kennesaw State University e-sports events such as “game on”. We will be using in-game mechanics such as command blocks that use in-game commands to change aspects of the game, and building to make a suitable map for our game. We will also be using custom coding in Java through plugins for the minecraft server. We need to make it lightweight so it is accessible to as many students as possible.

Project Website

<https://minecraft-turf-wars.netlify.app>

Deliverables

The deliverables for this project include:

- Source code
- Website/Github link
- Playable Minecraft minigame
- Final Presentation
- Final Report

Group Meeting Schedule Date/Time

Meeting within Microsoft teams every other Friday at 2pm, as per the kickoff meeting.

Collaboration and Communication Plan

Meeting on Discord. Meet before every deliverable, aiming for once a week at least.

Project Schedule and Task Planning

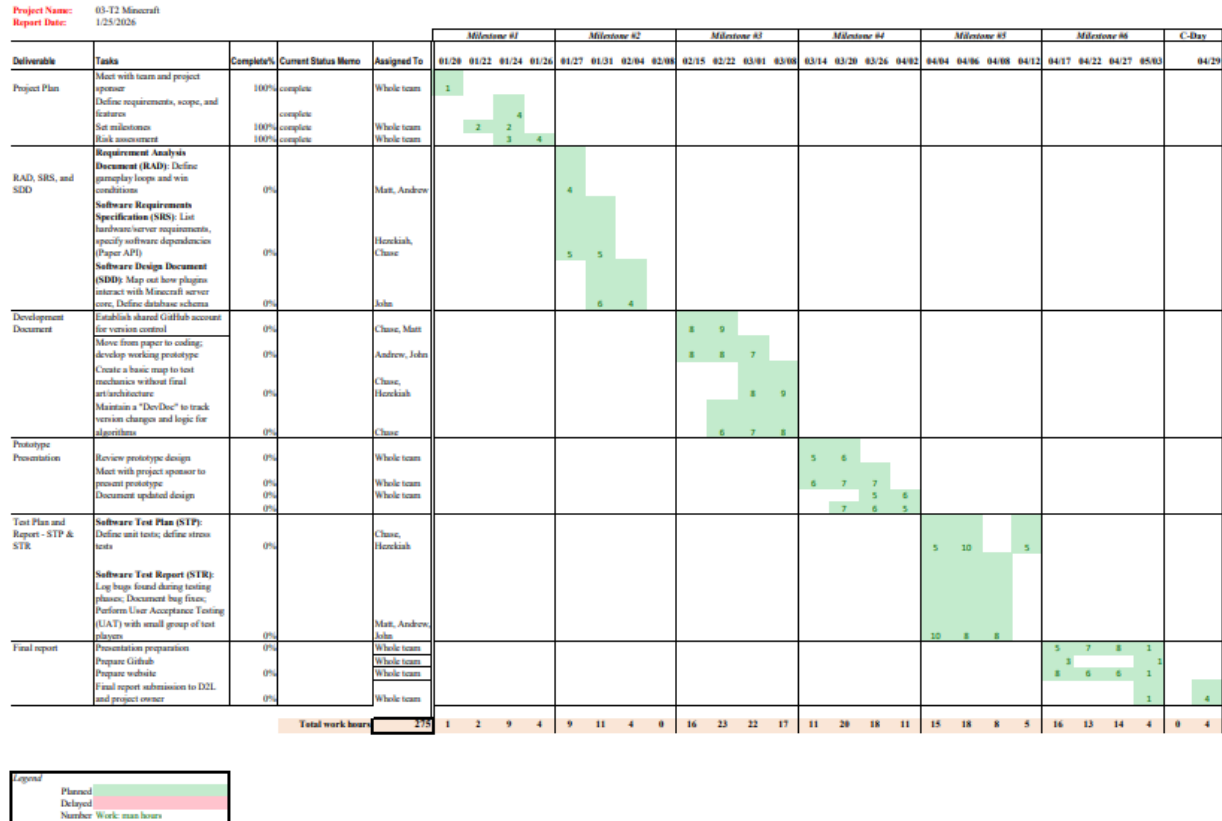


Figure 1. Gantt Chart

Other Plans

Risk Assessment for our project include:

- Bugs/crashes within game mode
- Server lag/Performance issues
- Players bypassing game boundaries
- Player data not saving correctly

Plans to help mitigate risks:

- Test one new feature at a time, assuring it works properly
- Adding error handling to code, preventing unexpected inputs
- Full game testing from start up, gameplay, to closing of game
- Test load with real players
- Teleport players back if leaving playable areas/restrict commands during game
- Save player data at key events

Version Control Plan

Access-

Will be hosted on Github within the main branch.

Branching-

Will be using a feature-branch workflow such as feature name, bugfix, etc.

Pull requests-

Every change will have a pull request which will state what changed, why, and how was it tested.

Releases-

All builds will be named v0.1.0, etc.

Documentation-

The repository will have a README.md, document folder for rules, design decisions, and troubleshooting, and a CHANGELOG.md for changes by the version.

Software Requirements Specification (SRS)

Overview

The turf wars minigame will be a plugin that is designed for the KSU esports minecraft server. The plugin will allow for players to join matches, be assigned to a team, and switch between a build phase and capture phase to gain territory in an arena. The arena size will be based on the amount of players competing.

Project Goals

A functional turf wars minigame mode that can be played in varying team sizes, assign teams that are balanced, and keep track of turf claims.

Definitions and Acronyms

PaperMC: a Spigot server implementation.

Plugin: a server-side add-on to extend minecraft.

Assumptions

The server will run a supported PaperMC version. The players will have a stable network connection to play on the server.

Design Constraints

The functionality of the gamemode will be limited by the Paper API and the way in which it "sandboxes" plugins and contains their access. Our implementation will be entirely contained within a Paper API plugin, which manages our minigame's I/O access and connection to the rest of the KSU Minecraft server.

Environment

The end product will have to be a portable, sandboxed Java module that does not depend strongly on outside information or make external I/O requests.

User Characteristics

Users will connect to a lobby subserver, from which they will be routed into instances of the arena world to play the game.

Functional Requirements

Game State & Lobby Management

- **Arena:** The game will take place in a defined world(s) where any changes do not affect the main server lobby, or any of the other game modes already present on the server.
- **Queue System:** Players will be able to join a lobby via a portal system that will transport them to a "queue world." Inside of the queue world players will wait for enough players (4-16 players) to start the game. The queue world will also have a parkour mini game implementation to give players an activity while they wait.
- **Auto-Reset:** Upon completion of the game the arena(s) must automatically revert to its original state (removing all player-placed blocks and resetting the turf line). Also, it must return the players back to the server lobby.

Turf Game Mode Mechanics

- **"Turf" Tracking:** The system must track the divide line between the two teams.
- **Kill-Based Advancement:** When a player kills an opponent, the divide line will move X blocks into the enemy territory.
- **Build Restrictions:** Players must be restricted to placing blocks only in their teams "turf," and not being able to place blocks on the enemy team's "turf."
- **Win Condition:** The game ends when one of the team's territory reaches 100% of the arena area, or whichever team has the most area controlled when the timer runs out.

Combat Mechanics

- **One-Shot Kills:** The system should register any arrow hit as an instant kill and begin respawning the player.

- **Respawn Mechanics:** The respawn mechanics will include a dynamic timer that is less for smaller games (2v2, 4v4, etc.) and greater for larger games (8v8, 16v16, etc.).
- **Materials and Ammo:** Each player should spawn in with 16 blocks and 16 arrows. During a predetermined interval of time (e.g. every 15 seconds) each player will receive an additional block and arrow.
- **Arrow Retrieval:** The system should automatically clear arrows from the ground to prevent lag.

Team Balancing

- **Auto-Balancing:** The system must distribute players into two teams Gold Team and Black Team ensuring even numbers at the start of the match.
- **Friend Queuing:** The system should prioritize two or more players queuing together being placed on the same team.
- **Spawn Protection:** Players should be invincible for 2-3 seconds after respawning to prevent "spawn camping."

Non-Functional Requirements

Security

- All players must not be able to use cheats or commands for gameplay advantages
- Players must not be able to glitch out of designated play area
- Only the Admin must have power to start, stop, or reset the game

Capacity

- The game must be able to hold a capacity of 32 players
- The game must support all entities without lag
- The game must support player action responsibly at an average speed of 100 ms

Usability

- Players must understand how to play game by start time
- Game must display time players have remaining in rounds
- Game must display win/lost result on screen
- Game must display the size of the lobby/teams

Other

- The game must get through playable rounds even when errors accrue
- The game must not lose player data when they disconnect from game
- The game must not crash when player disconnects in the middle of game round

Software Design Document (SDD)

Design Considerations

This gamemode is for use in Kennesaw State E-sports events. That means it must accommodate various low-high end hardware, as well as handle a large capacity of players. It also must be easy to understand and play for first time attendees of these events.

Architectural Strategies

Programming Language

- **Language:** Java 21+

PaperMC

- The industry standard software for running high performance Minecraft servers.

Future Plans

- Future game mode implementations for the Minecraft server

System Architecture

- The System will be divided into six subsystems:
- An area manager that will handle the creation, loading, and saving of the areas as well as player spawn locations.
- A match engine which will allow for control over each match's lifecycle and timing.
- A team manager which will assign players into one of two teams and make sure the teams are balanced.
- A turf manager to calculate territory ownership and will update when conditions are met.
- A rules manager which will set arena boundaries and gameplay restrictions.
- A UI manager which will display game information like the scoreboard, time remaining for each phase, and turf percentage.

Detailed System Design

The implementation of the turf wars minigame will largely be in the form of a Paper API plugin. The final deliverable will be a portable .jar file, created through Paper's Gradle build system, to be placed in the KSU Minecraft server's plugins directory. Existing infrastructure and the Paper API will take our code and host it as a playable minigame from there.

Classification

The Paper API's concept of a "plugin" is a modular Java package meant to be loaded on top of an existing Minecraft server. Plugins do not redefine any of the basic game logic associated with Minecraft, as that processing is handled by the game server. They use the Paper API to communicate with the game server to change game data, such as items and blocks in the world and players' inventories.

Constraints

The minigame will be online at only certain event times, as instructed by the people running the server. It should only need to store and gather state for as long as the minigame is online - any more persistent data like usernames will be handled by the API and the existing server. We may want to store state such as score records, which can be made to persist on server storage, but permanent storage will not be needed or considered as a functional requirement. The plugin component will likely have its own txt or yaml file for storing data, with I/O access moderated by the Paper API.

Resources

In order to function, the plugin will need the KSU Minecraft server to be up and running and ready to receive it. This implies a certain amount of CPU and RAM already allocated to the server, which is usually a minimum of 4GB and multiple processing cores. Given that the plugin will be running at times where server load is higher, the server will likely need more than this in order to provide adequate performance. The plugin itself will require minimal overhead in terms of resources, needing only enough storage and memory to load the lobby and arena worlds.

Glossary

Turf Wars - A minigame design within Minecraft involving two opposing teams that try and gain territory by getting eliminations using a bow and arrow.

Paper API (PaperMC) - High-performance Minecraft server API (application performance interface) used to create plugins and manage gameplay features.

Plugin - A Java component loaded by a Minecraft server to extend or modify the game functionality without changing core game code.

CPU (Central Processing Unit) - Fetches, decodes, and executes instructions to process data and control system operations.

RAM (Random Access Memory) - High-Speed, short-term memory used to temporarily store active data.

YAML File - A human-readable configuration file format used to store plugin settings and temporary game data.

Development Document

Implementation Detail Narrative

System Integration & Workflow Overview

The gamemode comes in the form of a Paper API plugin - a portable executable that is loaded into a modified version of the Minecraft server software at runtime. Internally, various classes and containers organize the lifecycle of loading/unloading the game worlds, as well as connecting and redistributing players.

As the plugin loads in at server startup, it calls the TurfWars class' onEnable() method, which runs startup operations, and then creates the ArenaManager and the GameManager. These are singleton objects that handle the loading of world files and the logical lifecycle of matches respectively. At present, the plugin creates two empty games at startup, and then waits for a player to use the /join [name] command to join one of them, at which point they are teleported into that respective world. When the game ends or the server shuts down, that game's copy of the world is unloaded, and all players are teleported back to the lobby world

Arena Manager

Conceptual Overview: A singleton class that handles all I/O operations for the server. It is instantiated on plugin startup, and then is called upon whenever a new world needs to be created. To do so, it copies the template world into a new directory, and then loads that into a new World object (provided by the Paper API).

Development Process: Began by implementing these functions in the "main" TurfWars class, but refactored to be its own object for separation of concerns.

Game Manager

Conceptual Overview: Another singleton class that stores a HashMap of Game objects, and has methods to create and delete matches of the turf war gamemode. The Game class itself talks to ArenaManager to request file operations when needed.

Development Process: Was originally part of the monolithic ArenaManager class, but split off again for more separation of concerns.

Database and External Connections

- The plugin does not connect to any external or database or web API, rather handling things entirely through local file storage and code. The Paper API provides methods for accessing the relevant directories assigned to the plugin's container, while files and

folders themselves are moved using the Java File class and its built-in methods. Everything is contained within the plugin's data folder, which is located at the minecraft server's root/plugins/TurfWars

Project Setup and Environment Configuration

This section will include the steps for setting up the project in a developmental environment, and for end users how to set up the project to play the minigame.

Prerequisites

- Minecraft version 1.20.1-1.21.11
- Java Development Kit (JDK) version 17 or higher
- Java version 21
- Integrated Development Environment (IDE) capable of running Java
- Git
- Gradle

Installation Steps

For Developers

1. Clone the repository (<https://github.com/minecraft-turf-wars/minecraft-turf-wars>)
2. Install dependencies
3. Configure environment variables
4. Insert the command `./gradlew build` into the command terminal

For Users

1. Install an instance of Minecraft version 1.20.1-1.21.11
2. Connect to the Kennesaw State University Esports Minecraft Server (mc.esports.kennesaw.edu)
3. Walk through corresponding portal to be connected to pre-game lobby
4. Have Fun!

Verification

Developers can confirm the project built successfully via the terminal response showing a successful gradle build, or confirming the .bat file was formed. **Users** can verify a successful build of the project if they are loaded into a pregame lobby queue and eventually placed into a populated game.

Game Design Document

Game Overview

Turf Wars is a fast paced, team based, player versus player experience that combines elements of a traditional “tug-of-war” game with Minecraft’s building and combat mechanics. Gold Team and Black Team face off in an arena where the ground they stand on represents their score in the game. The objective is simple: expand your team’s territory by eliminating enemy opponents.

Kennesaw State University Esports needed a fresh and engaging Minecraft gamemode for campus events. This game will satisfy all of these requirements helping drive more players to the KSU Minecraft server. This gamemode was purposefully designed with simplicity in mind so experienced players and new players will be able to enjoy the game together.

Gameplay

Objectives

The objective of the game is to claim the enemy territory and eventually claim the entire map. This is done by eliminating players on the enemy team.

Game Progression and Play Flow

The player will start in a build phase where they can build structures to act as cover. After the build phase ends there will be a capture phase where the teams will now eliminate players of the opposite team in order to claim a section of the arena. This phase will continue until a team is able to eliminate enough players that allows for the capture of the whole arena.

Mission Challenge or Puzzle Structure

There will be two phases; a build and capture phase. Once the build phase is complete the capture phase will start and will end when all the territory is captured or the timer runs out.

Game World

The Game World design really just consists of the waiting lobby and the gameplay map itself. The waiting lobby will be a medium sized space where players can move around freely, given the boundaries provided. In addition there will be some small obstacles/interactive objects that a player can utilize while they pass time. For user insight, information like player lobby size, time elapsed, and game mode selection will be displayed.

The gameplay map is designed to have two spawn locations for the opposing teams. This will be a room closed off from all directions facing away from the opposing team. Once the allotted time for game start has depleted, players will be able to move into the main area of the map. The main area resembles a relatively large rectangular field that connects the two spawn rooms. This field is where all gameplay takes place. The flooring will visibly be split in half by the two team colors, this will also be the visual cue for which team has the upper-hand. Other than about one or two climbable objects on each team's side, this area will mostly be empty. This allows for the players to have space to build blocks for coverage when given time to do so. This gameplay map is reset and used again after each playable round.

Characters and Opponents

- **Player Avatar:** Players will be able to use the skins/accessories owned by their microsoft account.
- **Characters in the Game:** Fully user based game, meaning the only characters in the game are the users on each opposing team.
- **Opponents:** The opposing team of users.

Levels

- One playable level/map is provided.
- Setup in a two team, Teamdeath match style of play
- All game mechanics will be provided and understood before game start
- Player progress is assessed as a team, as they get eliminations on the opposing team they will gain a color row corresponding to team color.

Deployment

- The gamemode is playable on minecraft on the official KSU minecraft server.
- Minimum requirements for running Minecraft java edition according to minecraft.net are: 64-bit Windows 10/11 OS, Intel Core i3-3210/AMD A8-7600 CPU, 4GB RAM, and Intel HD Graphics 4000 or NVIDIA GeForce 400 series GPU.
- Since this is online multiplayer, an ethernet connection is preferred. If this is not possible, strong Wi-Fi will work as well.
- To get started players must connect to the server and navigate the player to the portal labeled Turf Wars, there they will be taken to the lobby to wait for more players to start.
- After the lobby phase is over (sufficient players join, or host starts), all players in the lobby will be assigned to teams (chosen or random) and teleported into the map where the game will begin.

Development

- Minecraft/Paper API 1.21.11
- The code for our project will be in our team's public github repository.
- WorldEdit API

Test Plan and Report

Overview

Purpose

The purpose of this document is to define:

1. How the team will test the new gamemode
2. How the team will test server deployment of the gamemode
3. How we will test gamerules and mechanics
4. The basis of the test estimates
5. Any risks, issues, assumptions and test dependencies
6. The test schedule and major milestones
7. The test deliverables

Scope

This document details the testing that will be performed by the project team for the Minecraft project. It defines the overall testing requirements and provides an integrated view of the project test activities. Its purpose is to document:

- We will be testing the gamemode and gamerules and how they deploy to the server.
- We will first test locally independent of the server to test gamerules and find bugs.
- Then we will test on a server with multiple clients.

Scope of Testing

In Scope

Testing server and gamemode are running properly with multiple clients connecting.

Testing lobby functions (connecting, starting, etc.)

Testing gamerules are working (players cannot cross into the opposing side of the arena, one arrow kills, blocks and arrows are received periodically, etc.)

Testing gamerules bounds (is the arrow Interval too frequent, what about blocks?) I.e. are these mechanics as fun as they can be?

Out of Scope

Minecraft mechanics. Our game is built in Minecraft, we are not Minecraft developers and cannot change the inherent gameplay.

Wider KSU server architecture. The main hub, and other areas of the KSU playable server are not ours to change/test.

Test Cases

Test Case ID	Description	Expected Result
Turf Tracking	Tracks the divide line between both teams.	Tracks the divide line and updates when a team gets a kill.
Kill-Based Advancement	When a player kills an opponent, the divide line will move 1 block into enemy territory.	The territory will update once a player is killed by moving one block into the corresponding territory.
Build Restrictions	Players are restricted when placing blocks.	Players will only be able to place blocks within their territory and not one the enemy team's.
Win Condition	The condition needed to end/win the game.	The game will end once a team claims 100% of the enemy team's territory/ game arena
One-shot Kills	The combat mechanics for the players.	Any arrow hit will be counted as an instant kill and will immediately respawn the player upon a hit/kill.
Respawn mechanics	Players will respawn when killed/hit by an arrow.	After a player is eliminated the will immediately respawn within their appropriate team spawn area.
Materials/Ammo	Players will be provided blocks and a bow and arrow.	Players will be given blocks whose color corresponds to the appropriate team and a bow and arrow and will gain more arrows and blocks after 5 seconds pass.

Figure 2. Test Cases

Test Procedures

Test case	Steps
Turf Tracking	1. Start the match 2. Observe the divide line 3. Kill a player for both teams 4. Observe turf line progress and scoreboard
Kill-Based Advancement	1. Start the match 2. Kill players on both sides 3. Observe turf line progress and scoreboard
Build Restrictions	1. Start the match 2. Attempt to place blocks on the enemy team's side
Win Condition	1. Start the match 2. Repeatedly kill players until the line reaches the endpoint 3. Observe if the game correctly ends
One-shot Kills	1. Start the match 2. Shoot a player with a bow 3. Observe if player correctly dies in one shot
Respawn mechanics	1. Start the match 2. Have other player shoot you with a bow 3. Observe whether you respawn instantly after death and skip Minecraft's respawn screen
Materials/Ammo	1. Start the match 2. Deplete blocks and arrows by placing and shooting them respectively 3. Wait and see if the in-game clock refills these resources periodically as expected

Figure 3. Test Procedures

Test Environment

- Minecraft version 1.20.1-1.21.11

- Pre existing Minecraft server (Oracle Linux 8 ARM VPS)
- IDE capable of running Java
- Java Development Kit (JDK) version 17 or higher
- Java version 21
- Gradle

Software Test Report (STR)

- Record results for each test case
- Indicate Pass/Fail and Severity (e.g., Minor, Moderate, Critical)
- Simple table format is fine – recall our example from Requirements

Requirement	Pass	Fail	Severity
Turf Tracking	☒	☐	
Kill-Based Advancement	☒	☐	
Build Restrictions	☒	☐	
Win Condition	☒	☐	
One-shot Kills	☒	☐	
Respawn mechanics	☒	☐	
Materials/Ammo	☒	☐	

Figure 4. Software Test Report

Version Control

The project utilizes Git for version control, with the repository hosted on GitHub. This allows all team members to collaborate efficiently while maintaining a stable and traceable codebase.

A feature-branch workflow was implemented to organize development. Each new feature, bug fix, or improvement was developed in its own branch (feature/turf-tracking, bugfix/respawn-issue) before being merged into the main branch. This ensured that incomplete or unstable code did not affect the production version.

All changes were submitted through pull requests. Each pull request included a description of what was changed, the reason for the change, and how it was tested. This process enabled peer review and helped maintain code quality across the project.

Versioning followed a semantic format (v0.1.0, v1.0.0), allowing the team to track progress and identify stable releases. Major updates, minor improvements, and bug fixes were clearly distinguished through version numbers.

Additionally, the repository includes documentation such as a README.md file for setup instructions, a CHANGELOG.md to track updates across versions, and a dedicated documentation folder containing design decisions, rules, and troubleshooting information. This ensures the project is maintainable and accessible for future development.

Conclusion or Summary

The Turf Wars project successfully delivered a fully functional and engaging Minecraft minigame designed for use in Kennesaw State University esports events. The team met the primary objective of creating a lightweight, accessible, and enjoyable experience that accommodates both new and experienced players.

Throughout development, the team implemented core gameplay mechanics including team balancing, turf tracking, combat systems, and automated match management. These features were designed with performance and usability in mind, ensuring smooth gameplay even with multiple players on the server.

In addition to gameplay functionality, the project emphasized software engineering best practices such as modular system design, structured testing, and effective version control. The use of a PaperMC plugin architecture allowed the system to remain scalable, portable, and easy to integrate into the existing Minecraft server environment.

Testing confirmed that the majority of core features perform as expected, with systems such as kill-based advancement, respawn mechanics, and build restrictions functioning reliably. Any remaining issues are minor and can be addressed in future iterations.

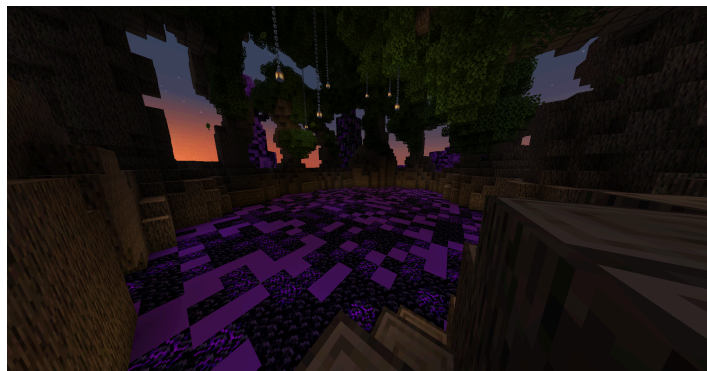
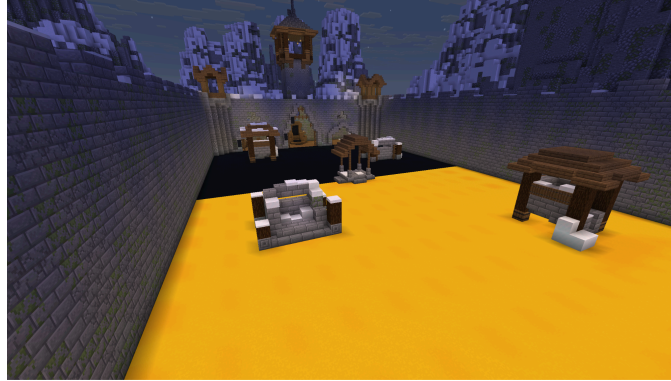
Overall, the project demonstrates a successful combination of game design and software development. Turf Wars provides a strong foundation for future enhancements, including additional game modes, maps, and expanded features, making it a valuable addition to the KSU esports Minecraft server.

Appendix –Training Verification

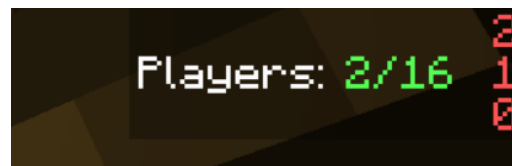
The appendix contains supporting materials that verify the functionality, testing, and completion of the Turf Wars project.

Included Materials:

- Screenshots of the game lobby and active gameplay sessions



- Evidence of successful server deployment and player connections



- Test results demonstrating core mechanics (turf tracking, combat, respawn system)



<https://minecraft-turf-wars.netlify.app/#demo>

These materials serve as validation that the system operates as intended and meets the requirements outlined in the Software Requirements Specification (SRS) and Test Plan.

All appendix materials are available in the project repository and can be accessed through the following link:

<https://github.com/minecraft-turf-wars/minecraft-turf-wars>