**GameMaster  
Design Document**

**CIS 3296 Section 705**

**Spring 2022**

**Team Members:**

Kian Azadi Justin Gallagher

Cameron Metzinger Jay Newman

**Repository URL:**

https://github.com/cis3296f22/GameMaster

Table of Contents

[Project Proposal 3](#_Toc120693261)

[Project Abstract 3](#_Toc120693262)

[High Level Requirement 3](#_Toc120693263)

[Conceptual Design 3](#_Toc120693264)

[Proof of Concept 3](#_Toc120693265)

[Background 3](#_Toc120693266)

[Required Resources 3](#_Toc120693267)

[Project Design 4](#_Toc120693268)

[Vision 4](#_Toc120693269)

[User Personas 4](#_Toc120693270)

[Class Diagram 6](#_Toc120693271)

[Sequence Diagram 9](#_Toc120693272)

[API Documentation 11](#_Toc120693273)

[Framework 12](#_Toc120693274)

[Testing Output 12](#_Toc120693275)

[Coverage Report 13](#_Toc120693276)

[Project Progress 14](#_Toc120693277)

[Week 2 Progress 14](#_Toc120693278)

[Week 3 Progress 15](#_Toc120693279)

[Week 4 Progress 16](#_Toc120693280)

[Week 5 Progress 17](#_Toc120693281)

## Project Proposal

### Project Abstract

This document proposes a discord bot application that serves as a hub for users to play small board games. The discord bot will be added to a discord server and the games will be played within a discord server. Users can customize visually aspects of the game as well as board size. GameMaster will allow discord users to connect and have fun through playing games.

### High Level Requirement

GameMaster is a fun minigame bot that allows users to play multiple minigames, either alone or with friends!

### Conceptual Design

The initial design concept is to have a discord bot created and added to a server. Have users in that server become authorized with the bot and communicate to each other through the bot. This will then be modified to having the bot send game information back and forth between the users.  
This project will use JavaScript programming language, the Discord API, Nodemon to run the project, and OAuth2 to authorize the users with the discord bot.

### Proof of Concept

<https://github.com/jay-newman/newBot>

Above is a link to a discord bot with reply commands to send back images and gifs. In this project you will find a readme file that explains how to set up and run the discord bot. This software was run on Linux Mint 20.3 Cinnamon. Version 5.2.7 using Javascript/node version 18.9.0.

### Background

The initial design will use emojis allowing the discord bot to print the checkers board. This was seen in a similar project at <https://github.com/jmsheff/discord-checkers> . This project aims to have a cleaner looking checkers board. Other ways will be explored into which this is possible and would allow the users to customize the game board and pieces instead of using the same colors of red and black.. In this project there will be the ability to change some rules of the common game of checkers. This can be done through the bot which will use different logic for the different game styles. This is the initial approach but expanding beyond to different games is another goal that will allow users to play other games besides just checkers. This result is intended to be a fun game playing discord bot that will allow users to play games with friends with a customizable experience.

### Required Resources

There are many resources available to develop a discord bot using javascript. There are written tutorials and videos that will be used in order to get a bot up and running. Software used will be IntelliJ IDEA. Any modern computer/phone with discord app will work with discord so hardware resources are not limited in any way.

## Project Design

### Vision

FOR Discord users WHO want to have a fun minigame to interact with friends on the platform, THE GameMaster is a Discord bot THAT will help users set up small minigames, such as Tic-Tac-Toe.

UNLIKE other Discord bots, such as Tic-Tac-Toe bot, the focus of GameMaster is to provide a "hub" for users to have multiple games of varying difficulty and engagement. OUR product acts as a "one stop shop" for different fast and fun minigames between friends on Discord.

### User Personas

Brandon, gamer that works from home

Brandon, age 24 works a sales position for a payroll company. He used to work in the office but when the pandemic happened he had to work from home. In his spare time he loves to play video games. Brandon would rather play video games and talk with his friends on discord then make endless cold call after cold call for work. Brandon has been able to be productive at work while still playing a lot of video games daily.

He found himself getting very bored of all the video games that he was playing. He wishes there was an easy way for him and his discord friends to play quick simple games that are fun to pass the time. He is looking for games with low learning curves so his friend will be more tempted to play with him. Brandon and his friends already all use discord to connect so it would be very convenient if there was a way to play games through discord. This would allow him an easy outlet to pass the time when he gets bored.

Corey, a Human Resources Department Manager

Corey is a 35 year old Human Resources Manager for a technology department at a large corporation. He was just assigned to a team of 20 individuals that all work a variety of roles within the department. Some of the people who work in his department work from home and others may not interact much due to their work not needing the communication. Corey would like to get the whole team together to help cultivate a more cohesive and communicative team. Unfortunately, he faces too many limitations to make that happen in person. He believes that he found the right platform to allow for communication and comeraderie in the GameMaster bot on Discord. He can create a private server for his team and they can all get to know each other better through playing games without too much effort.

Joey, a tech savvy junior looking to connect with family and friends.

Joey is a young teen looking into hosting his own Discord server for friends and family. While looking for some fellow gamers to join up in group games, he discovered that using Discord was an incredibly fun and safe place to be and meet people with shared interests. After speaking with his family about creating a server so that they could stay in touch, Joey’s parents gave him permission to start his own server and moderate it. He is looking forward to including small/quick games into the server to stay connected with his family and friends.

Theodore, a kind-hearted dentist who wants to spend time with his grandchildren

Theodore, or "Ted" for short, is an 87 year old grandfather of 5 who watns to spend more time with his grandchildren, who live throughout the United States. He heard them discussing Discord prior as a way to keep in touch, and successfully created his own server he can use to keep in touch which all of his grandchildren at once. A huge fan of quick and easy games to play, such as Tic Tac Toe, Ted downloaded GameMaster to join his server so he can have quick and easy games to play with his grandchildren whenever they all have time in their busy schedule. This way they can always be in touch even outside of major holidays.

### Class Diagram

Diagram

Description automatically generated

**Diagram 1 –** UML class diagram focusing on the aggregation of the TicTacToe class. The isolation of only aggregations allows for a clean understanding of this particular relationship, so as to simplify other future diagrams showing other relationships.

**Diagram 1** focuses on the aggregational relationships seen in TicTacToe. The program starts by creating a TicTacToe object, which then creates EventHandler and TicTacToeBot object. This EventHandler object is also used within the TicTacToeBot. The TicTacToeBot object then has a cascade of relationships, starting by creating a GameCommand object. This object creates a GameStateManager object, which in turn utilizes a GameStateValidator object. These objects in **Diagram 1** allow for us to create the backbone of GameMaster, allowing for future dependencies to run effectively (see **Diagram 2** and **Diagram 3**).

Diagram

Description automatically generated

**Diagram 2 –** UML class diagram focusing on the dependencies and inheritance of these dependencies of the classes seen in **Diagram 1**. This is the first diagram of these dependencies, and GameBoard is purposefully left blank as they are the focus of the following diagram.

**Diagram 2** looks at the dependencies and inheritance of the objects seen in **Diagram 1** that allows for the program to actually prepare the bot. TicTacToe, which was previously described, utilizes the discordJS object that contains all code provided through the Discord framework. The TicTacToeBot object uses an AppCommandRegister object, which allows for the bot to be interacted with by the user. Note that although EventHandler is used by both TicTacToe and TicTacToeBot, it does not have any dependencies and as a result is not present in **Diagram 2**. GameCommand uses two MessagingTunnel objects, both of which extend the MessageTunnel object: CommandInteractionMessagingTunnel and TextMessagingTunnel. These both have similar functionality, however have different attributes and purposes. Finally, the GameStateManager class is where the majority of the logic occurs. This class contains a DuelRequest object, which uses discordJS and also has yet another extended MessagingTunnel in the form of ComponentInteractionMessagingTunnel. GameStateManager also contains the AI object, which is a fairly self-contained object relative to the othes. Note that GameStateManager also contains GameBoard, which is not characterized in this diagram. This is due to it being fairly elaborate objects, and they are explained further in **Diagram 3**.

Diagram, schematic

Description automatically generated

**Diagram 3 –** UML class diagram focusing on the relationships of GameBoard, the empty object seen in **Diagram 2**. Note this diagram contains all kinds of relationships, not specified to specific kinds like **Diagram 1** or **Diagram 2**.

As can be seen on **Diagram 3**, the GameBoard class is a sort of “central hub” to get the game board created. It is able to create a generic GameBoard using GameBoardBuilder, or one using buttons using GameBoardButtonBuilder. All of these are used in part by the Game object used by GameBoard, allowing for the actual gameplay of TicTacToe to take place within the program. Note that the AI object, while explained in **Diagram 2**, is used in GameBoard, and discordJS is also used in the GameBoardButtonBuilder class. The latter signifies the importance of the discord.js framework within the entirety of this program, as it has been found in all Diagrams to varying degrees.

### Sequence Diagram

Graphical user interface, application, timeline

Description automatically generated

**Diagram 4 –** UML sequence diagram illustrating how a player is able to challenge another to a round of TicTacToe.

**Diagram 4** illustrates a player challenging another player to a TicTacToe duel. PlayerOne uses slash command /tictactoe PlayerTwo to challenge to a game of TicTacToe. GameMaster recieves the slash command and instantiates the necessary objects to run a game of TicTacToe. TicTacToeBot then sends the duel request to PlayerTwo. PlayerTwo responds accepts the duel. This results in TicTacToeBot collecting messages from PlayerOne and PlayerTwo. The first move of the game is chosen at random. Each move is verified to make sure it is the correct player making the move. The board is updated after every move. After each player makes a move TicTacToeBot checks for a winner. If no winner is found the process is repeated until a player has won. Once a player has won each player is notified. TicTacToeBot and associated objects are then destroyed. If a new game wants to be played it will be initiated with GameMaster.

Graphical user interface, application

Description automatically generated

**Diagram 5 –** UML sequence diagram illustrating what happens when a duel request is sent to the AI rather than another player.

**Diagram 5** illustrates a player challenging the AI to a TicTacToe duel. PlayerOne uses slash command /tictactoe to create a game of TicTacToe. GameMaster recieves the slash command and instantiates the necessary objects to run a game of TicTacToe. TicTacToeBot then sends the duel request to the AI, which automatically accepts and starts the game. A starting player is chosen at random, and then that player gets to go first. If it is the player, they click the location they want to choose, and then TicTacToeBot checks to see if the move is valid. Once the move has been determined valid or invalid, TicTacToeBot will either have PlayerOne try again, or notify PlayerTwo (the AI) it is their turn, respectively. The AI will then see if there are any defensive moves to be made, followed by any offensive moves. If no moves are found, it will randomly select and available spot to fill. The turns go back and forth until a winner is determined.

Graphical user interface

Description automatically generated

**Diagram 6 –** UML sequence diagram illustrating how the commands are parsed by index.js.

**Diagram 6** illustrates a player using a slash command. The slash command is looked for in the slash command directory and executed. The game then begins. If embed message games are played than emoji reaction collection begins.

### API Documentation

Documentation was created by JSDoc 4.0.0, and can be found at the following link:

<https://htmlpreview.github.io/?https://github.com/cis3296f22/GameMaster/blob/main/Documents/JsDocs/index.html>

Testing

### Framework

The framework used for testing our bot was Jestjs (<https://jestjs.io/>), which is able to work with Node.js projects quickly and effectively.

### Testing Output

A screenshot of a computer

Description automatically generated with medium confidence

**Image 1 –** The output of the Jest testing suite. Note that as of now, only in-house code is being tested. This is due to the previously acquired code needing to be refactored and navigated more prior to writing successful tests. More tests are being added for both past and future implementations.

The tests in **Image 1** shows how the testing suite runs in VSCode, representing our ability to add tests as we develop them. As is stated in the image description, this is an incomplete representation of the tests available, which is more apparent in **Image 2**. The above image is meant to show how the testing output looks on the IDE.

Text

Description automatically generated

**Image 2 –** The output of the Jest testing suite done through the command line. This implies testing can be done through GitHub itself without needing to have the entire testing suite on a local machine.

The output shown in **Image 2** shows a more complete version of the available tests, which was focused on ensuring RPS was working as intended. This is due to RPS being the first in-house game fully developed independently, and as a result has code that does not need to be extensively refactored to allow for ease of testing.

### Coverage Report

Text

Description automatically generated

**Image 3** – Coverage report of the tests seen in **Image 2**. Not that due to the large volume of code present in this project, extensive testing will be needed to provide extensive coverage. This is a goal that is expected to be completed in the upcoming Sprint.

**Image 3** shows how the coverage report looks after the tests are run. As of now, there is relatively low coverage, but this will be fixed in the upcoming sprint as extensive testing is being implemented (not included now due to being a work in progress that does not have a meaningful change to current results). In addition, some of the coverage percentages are not entirely representative of the actual coverage, partially due to the formatting of the files being tested.

## Project Progress

### Week 2 Progress

**Sprint Goal:** The goal was to allow for greater customization of the currently available code through refactoring classes that have the highest reusability. This allows for the usage of these objects in more than just one scenario, opening up multiple possibilities in the future

**Backlog Features**

* Customization of current TicTacToe game
* Generalization of common components that can be used for future features

|  |  |  |
| --- | --- | --- |
| Tasks in Sprint | Task Status at end of Sprint | Assigned To |
| Fix current Issues | Completed | Justin, Jay |
| UML Diagram | Completed | Kian |
| Design Document updates | Completed | Kian, Jay |
| Board refactored | In Progress | Kian |
| TicTacToe refactored | In Progress | Kian |

### Week 3 Progress

**Sprint Goal:** The goal was to start adding new features to the bot, and to pave the way for the implementation of game variations, and new minigames altogether.

**Backlog Features**

* Customization of current TicTacToe game
* Additional minigames such as Connect4
* Deployment of the bot to a service like Heroku
* Refactoring of the AI class to support boards and games of different sizes.

|  |  |  |  |
| --- | --- | --- | --- |
| Tasks in Sprint | Task Status at end of Sprint | Assigned To | Velocity |
| Sequence Diagrams | Completed | Jay | 3 |
| Rock Paper Scissors | Completed | Jay | 8 |
| Design Document updates | Completed | Kian, Jay | 3 |
| TicTacToe refactoring | Completed | Kian | 8 |
| Bot deployment | Completed | Cameron | 5 |
| Handling Bug Reports | Completed | Justin | 5 |

**Estimated Velocity:** 32 points

**Effective Velocity:** 100% (32/32)

### Week 4 Progress

**Sprint Goal:** The goal was to add new minigames to the bot, refine the old minigames, and implement a unit testing framework.

**Backlog Features**

* Addition of new minigames
* Refactoring of the AI class to work on different-sized TicTacToe boards
* Release and maintenance of the Heroku server bot

|  |  |  |  |
| --- | --- | --- | --- |
| Tasks in Sprint | Task Status at end of Sprint | Assigned To | Velocity |
| Sequence Diagrams | Completed | Kian | 3 |
| Rock Paper Scissors Refactor | Completed | Jay | 3 |
| Design Document updates | Completed | Kian | 2 |
| AI refactoring | Completed | Kian | 5 |
| Release Updates | Completed | Jay | 2 |
| Unit Testing Setup | Completed | Justin | 3 |
| Individual Unit Testing | In Progress | Justin | 3 |
| TicTacToe Refactor | In Progress | Kian | 2 |
| Coverage Report | In Progress | Justin | 3 |
| Start Blackjack Minigame | Completed | Cameron | 3 |
| Hi-Lo Card Game | Completed | Jay | 5 |
| Class Diagram Updates | Completed | Kian | 2 |

**Estimated Velocity:** 36 points

**Effective Velocity:** 78% (28/36)

### Week 5 Progress

**Sprint Goal:** The goal was to wrap up currently existing projects and goals, and to provide a “fully baked” product for final release.

**Backlog Features**

* Addition of new minigames
* Release and maintenance of the Heroku server bot

|  |  |  |  |
| --- | --- | --- | --- |
| Tasks in Sprint | Task Status at end of Sprint | Assigned To | Velocity |
| Bot Displays Welcome | Completed | Jay | 2 |
| Rework timer for HiLo | Completed | Jay | 2 |
| Add record to RPS | Completed | Jay | 2 |
| Test HiLo | Completed | Justin | 2 |
| Test RPS | Completed | Justin | 2 |
| Reload Bot | Completed | Cameron, Jay, Justin | 1 |
| Project Presentation | Completed | All | 1 |
| JsDocs for TicTacToe | Completed | Kian | 5 |
| Refactor AI | Completed | Kian | 3 |
| Let User Choose Board Size | Incomplete | Kian | 2 |
| Test AI | Completed | Kian | 3 |
| Final Github Release | Completed | Jay | 2 |
| Final Design Document | Completed | Kian | 3 |

**Estimated Velocity:** 30 points

**Effective Velocity:** 93.33% (28/30)