RISK Bot Vs. Bot Website

Software Requirements

Specification

Individual Contributions

Dale Morris

Overall Description

Dale Morris wrote, researched, and finished the draft of the Overall Description section of the report.

Xavier Beatrice

Specific Requirements

Xavier Beatrice wrote, researched, and finished the Draft of the Specific Requirements section of the report.

Formatting

Xavier Beatrice formatted and designed the document so it looked appealing.

Editing

Xavier Beatrice edited and finalized the work throughout the report to prepare it for the final submission.

Louis Wiley

Introduction

Louis Wiley created the Introduction segment and the cover page for the document.

Introduction

Purpose

This documentation defines the requirements for developing a web server that allows students to create RISK AIs and pit them against each other. By accomplishing this project, we aim to understand better how AIs are made and how to run a server that takes and runs user software while still being responsive and performant. By the end of the project, the website will be hosted, and students will be able to interact with and learn from its resources.

Definitions

- AI: Artificial Intelligence refers to the simulation of human intelligence by machines, particularly computer systems, enabling them to perform tasks that typically require human cognition, such as learning, reasoning, and problem-solving.
- GUI: Graphical User Interface refers to a visual system that allows users to interact with software or devices through graphical elements like icons, buttons, and windows rather than text-based commands.
- XNA: Refers to a set of tools and runtime libraries developed by Microsoft to facilitate game development on Windows, Xbox, and other Microsoft platforms, providing a managed environment with built-in support for handling graphics, audio, input, and other game-related functionalities.

System Overview

For this project, we will need to create a version of RISK to integrate the AIs with. We will also need a website to host the game and demonstrate our program. Additionally, we will need a way for users to submit their own AIs to integrate with our game. Finally, we will need to develop our own AI that can beat a bot that only makes random decisions in a RISK game.

References

XNA Development Site

http://xnadevelopment.com/tutorials/gettingstartedwithxnadevelopment/GettingStartedWithXNADevelopment.shtml

Blazor Development site

https://dotnet.microsoft.com/en-us/learn/aspnet/blazor-tutorial/intro#:~:text=Step-by-step%20instructions%20for%20building%20your%20first%20Blazor%20app.

Overall Description

Product Perspective

System Interfaces

The Internet will be used as the interface between clients and the server.

User Interfaces

Our users' interface will be a website where they can choose to play the game themselves or submit a bot to play it.

Hardware Interfaces

Users will either play the game or submit a bot using a keyboard, mouse, and monitor.

Software Interfaces

When users submit bots to play Risk, they will be able to use an API written in one of several languages including C#, Java, and Python.

Communication Interfaces

Communication interfaces do not apply to this project because it does not involve communication.

Memory Constraints

Our goal is for our program to use at most 1GB of memory at any given moment. This will ensure the program will comfortably run on whatever server we use for our website.

Operations

The program will handle turn-by-turn actions input by users–human or robotic–while updating the game state until one user wins the game. Game logic will be handled by the server while displaying the GUI will be handled individually by clients.

Site Adaptation Requirements

Site adaptations do not apply to this project because users are expected to already have access to the hardware they need to interact with the program without adaptations.

Product Functions

The product will perform two key functions. The first function is to allow human users to play a digital version of the board game Risk including a GUI. The second function is to allow users to submit bots that can stand in for the user to play the game. This will involve creating an interface via which the bots will be able to make moves.

User Characteristics

The users will be split into two groups. The first group is general audiences who simply want to play Risk. The second group is made up of people who want to submit a bot to play Risk. The latter group will primarily be made up of students who are taking courses in artificial intelligence and who want to design an AI to gradually improve at the game. While both groups deserve consideration, the latter group is the primary target audience of our program and will merit more consideration than the former group.

Constraints, Assumptions, and Dependencies

Our primary constraint is money. We have no budget to work with, so any solution to this project must be free to produce, run, and maintain. In terms of assumptions, we will be assuming that if a user submits a bot to our program, it will be written in a language that is compatible with our software. This means it must be written in C#, Java, or Python, although granted enough time, this list of languages may expand. As far as dependencies go, our project depends on a couple of frameworks. Namely, we will use XNA for handling gameplay interactions between users and the game, and we will use Blazor to support our website.

Specific Requirements

External Interface Requirements

The main external interface we'll interact with is the end user; our web interface will be the primary external interface. It must support multiple users who interact with the upload and download systems simultaneously and be easily usable by the users. We'll also need to ensure that our service doesn't conflict with any common modern browsers.

Functional Requirements

The service we're creating must meet a few functional requirements. First, it must be able to provide a user with a random risk bot from the internal database. The service must also provide a graphical web interface that users can interact with, accept, and store user-created risk bots. The service must be able to provide the user with some method of viewing a game run on the server between 2 different user bots. The service must have a bot that is used as a fallback and can generally defeat a bot that only uses random moves.

Performance Requirements

The service should be able to support many users' simultaneous usage and have minimal bottlenecks that inhibit performance beyond what is necessary or innate to the system running the service. The service should require little to no maintenance to upkeep. The service should be frugal with its hardware and storage usage. The dotTrace .NET profiler should be used to enable further meeting of performance requirements.

Design Constraints

One key design constraint is cost and minimization, as the project has low funding. All key software should also be hosted and maintained on the same server and runnable through the same Docker setup script.

Standards Compliance

There are no notable standards that must be complied with.

Logical Database Requirement

The database will be fairly simple, containing only stored locations for the accepted users' RISK bots and the user who sent them in. The Key would be the user's provided username.

Software System Attributes

Reliability

The service should have no way for a user using the service as intended to cause a server hang or failure that restricts the user's ability to use the service as intended.

Availability

The service should be available at all times besides an occasional restart time that could occur regularly to clear cache and memory issues.

Security

The service should not be easily hackable or have any way for the user-provided code to run maliciously. The service will likely use virtual machines to perform this task.

Maintainability

The service should use well-maintained libraries and languages to ensure that future developers can continue the project after the original developers leave it.

Portability

The service should be highly portable by using a docker container to encapsulate the software.

Other Requirements

The service should be mostly hardware agnostic and not use any software or systems that would tie it directly to a specific cloud provider. It should also be able to run on dedicated hardware as necessary with little additional steps. Instructions should be provided in the service's codebase on how to create and set up the service so that maintainers or interested individuals in the future could run the service on a home machine or local server. The service should also be usable in some manner in a local environment so that users can test their RISK bots before uploading them to the server-hosted service.

Key Personnel

- Xavier Beatrice
  - Team-lead
  - Fill-in Developer
- Louis Wiley
  - Front-end Developer
- Dale Morris
  - Back-end Developer