roject Deliverables

To meet the proposed goals of this project, we must achieve the following deliverables in, roughly at least, the following order. We can't start writing code without first producing the basic game design, we can't select a development framework without first understanding the application we're creating, and we can't select the development tools without first selecting the framework we'll be working within.

Goal: Produce a client/server software solution that provides an engaging, challenging, and rewarding experience for pre/early teens centered around writing code in an instructor-led, collaborative team environment, supports self-paced learning and guided exploration of programming constructs, and introduces the concepts of AI/ML.

Objective 1: Design a Game System that's easy to learn, but hard to master.

Deliverable 1: **Game Design Document Created.**

- The following is a high-level outline to server as an example of how difficulty could scale, but specific rules, scoring, and other details will also be included in the final design document.
    - *Beginner Mazes* will consist of simple, 3x3 to 5x5 maze grids that will be very easy to beat, even after adding in the complexity of having to script an AI to navigate it on the player's behalf. The beginning mazes will lack traps, secret doors, and power-ups, but may provide bonus point opportunities in the form of treasure chests that improve the final score for players/teams that find them.

- *Intermediate Mazes* will introduce traps that emit warning indicators into surrounding rooms, giving the programmer ample opportunity to detect and avoid them. These mazes will provide more nuanced opportunities to earn bonus points and will also introduce secret doors and shortcuts that allow the observant programmer to script an AI that can complete the maze in fewer moves.
- *Advanced Mazes* will introduce undetectable traps and procedurally generated potions of varying positive and negative effects. This will require the programmer to implement a memory feature for their AI so that it can avoid undetectable traps and experiment with experiment potions of unknown effect so that it can drink only the beneficial potions during subsequent runs of an advanced maze.
- *Master Mazes* will be massive, sprawling labyrinths that introduce internal exits (that is, exits that are not on the edge of the maze).
- *The Final Challenge.* This maze will be kept secret until the last day of camp. It will be the largest, most complicated maze yet and will include multiple instances of all of the mechanics that the player's AIs have encountered already. The only twist is that there will be multiple exits - will the AI be smart enough to

notice? And if it does, will it be able to identify which exit provides the best final scoring opportunity?

Objective 2: Design and Implement the Game Server.

Deliverable 2: **Development Platform and Tool Stack Selected.**

- With the game design in place, we now have to decide which development framework and tools would help us implement that design most efficiently. Key considerations for a procedural, maze-based game of the type being proposed include memory management to support the recursive pathing algorithms needed to generate and solve mazes, support for the basic HTTP methods, support for text compression over HTTP for client/server communications, and support for data persistence via database connection objects or APIs.

Deliverable 3: Implement Procedural Maze Generation Service.

- Based on the game rules provided by Deliverable 1, we start by developing a stand-alone service that can generate mazes of varying difficulty using a seeded, pseudo-random number generator that allows us to regenerate mazes based on a seed value so that we can update the maze database during development should the rules need to be adjusted as we progress toward final release.

Deliverable 4: Implement Game Server, Game API, Team Service, and Score Service.

- *The Game Server* will handle single-player and team-based game

sessions by creating instances of mazes at the request of the client application, tracking player/team progression through the selected maze, and reporting outcomes and final scores to the Team and/or Score service when the game is completed, abandoned, or has timed out.

- *The Game API* will be baked into the Game Server, providing RESTful endpoints that allow the client application to interact with an active game session by issuing commands like "move north," "jump south," and "drink potion." The game API will return textual responses representing the outcome of issued commands that include phrases like "You see exits to the north and south," "The potion tastes like snail slime. You quickly start to feel very slow," or "There is a horrible sound that reminds you of a meat grinder coming from the room to the south."
- *The Team Service* will manage team-specific information and maintain a relationship to team-based scores managed by the Score Service. It will also associate individual campers with the specific bots that they control within a team.
- *The Score Service* will provide access to sortable, searchable score data and maintain records of each team's trophies and achievements. The Score Service will be manually accessible, but will primarily be used to provide data to the Scoreboard application.

Milestone 1: The Maze Game Is Now Playable!

- The game can now be played, but without a client application game commands have to be sent to the server using tools like *curl* or *PostMan*.

Objective 3: Implement the Client Application.

Deliverable 5: Student Scripting Language Selected.

- The student's interaction with the game will be restricted to using the game's API provided by the game server, so the scripting language must support the the same protocol and data transfer requirements that the game server uses. This is the language that students will taught, so it must be something freely available and wildly popular with tremendous amounts of documentation and community support.

Deliverable 6: Client Application Implemented.

- The client application will be a very simple, lightweight shell that encapsulates communication with the game server. A client application instance will be created for each team in camp, and each member of each team will be assigned a discrete script file in which they will write the code for their individual bots. The client application will support both single-bot and full-team game play modes to allow students to work independently and at their own pace, though scores, trophies, and achievements will only be recorded during team play.

Milestone 2: Game Supports Scripted Bots!

Objective 4: Design and Implement Configurable Scoreboard

Deliverable 7: Scoreboard Application Implemented.

- This standalone, web-based scoreboard application will data from the score service to provide public a ranking of team-based games completed so far. The scoreboard is intended to encourage collaboration and team-based gameplay by leveraging the competitive nature of many pre/early teen gamers to focus attention and drive innovation. Unfortunately, such a scoreboard can have the opposite effect because publicly displaying team rankings can, in some situations, result in frustration, disengagement, demotivation, and possibly even anger or hostility among young gamers. To mitigate this risk, the scoreboard application must include a feature that allows the instructor to quickly disable it should it's use prove counterproductive.