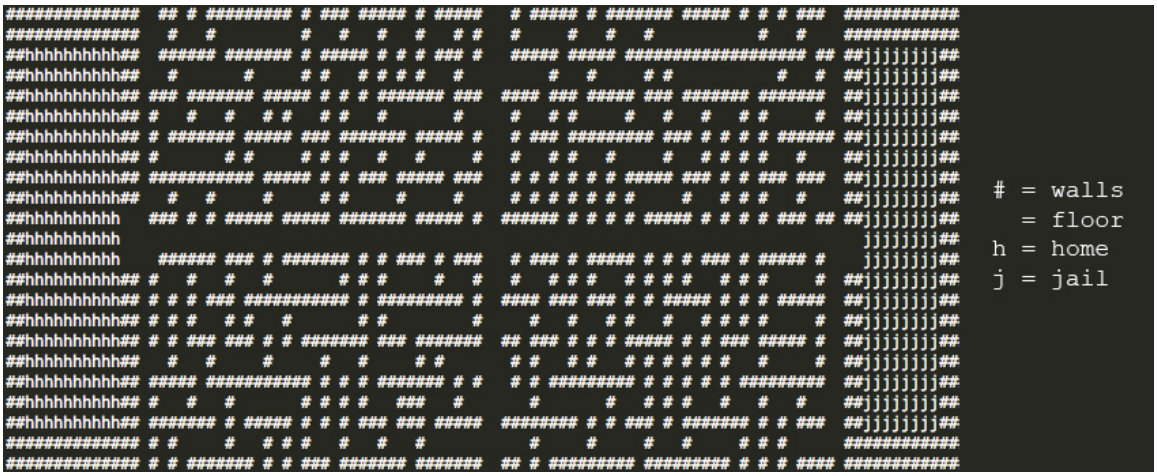
Capture the Flag System Specification

Team BSODS
Ethan DuBois • Niko Konstantakos • Eddie Rivas
CS 451 • Distributed Systems

Specification

1. Types of maze cells and states associated with them

There are four kinds of cells in our game: floor cells, wall cells, home cells and jail cells. The image below shows a section of the map with the different cell types labeled.



• **floor cells (blank spaces):** floor cells correspond to the spaces available for a player to move into. Few random floor cells in the map may contain a **jackhammer** item (for more details, read **Jackhammer** under the rules section) available for pick up. Floor cells may be **free**, **occupied**, or **containing an item**.
• **wall cells (# spaces):** wall cells correspond to spaces a player can not move into and comprise of the maze players must make their way through. Wall cells (except for the outer-most walls) can be removed and turned into floor cells permanently by a player using a **jackhammer**.
• **home cells (h spaces):** home cells make up the "home" area where players of the same team spawn together at the beginning of the game. A team's own flag will spawn near this area.
• **jail cells (j spaces):** jail cells make up a team's "jail" area, where players on the opposite team that get tagged will be moved to and confined until another teammate liberates them.

2. States and attributes of a player

States: players can be in any of the following states based on different conditions. These are special states the player takes on, a regular active player is not considered to be in any special of state.

• **jailed:** a jailed player is one who's been tagged by an opponent and placed in the opposite team's jailed. In this state, a player can not leave the jail until a teammate tags them and removes the status.

• **has_flag:** a player in this state has picked up and is carrying the opposite team's **flag**. In this state, a player is vulnerable to tagging even in their own side and cannot pick up a **jackhammer** unless he/she drops the **flag**.
• **has_jackhammer:** a player in this state has picked up and is carrying the **jackhammer** item. In this state, that player may face a wall and press the corresponding key (tbd) to destroy a wall cell, effectively turning it into a floor cell. The amount of times the **jackhammer** may be used is o be determined.

Attributes: these are the different player-variables that determine the parameters for the player.

• **`location`:** an <x,y> coordinate that determines the current cell the player is occupying.
• **`direction`:** this determines the direction (up, right, down, left) which the player is currently facing, determined by the direction last moved. This is used to determine which wall the player will break when using the **jackhammer**.
• **`team`:** an integer value that determines the team the player corresponds to, which governs several of the game's factors, such as side, home, jail, opponents, etc.
• **`player_no`:** a player's distinct player number, making it the player's personal ID.
• **`session_no`:** a reference to a player's connection (file descriptor).

3. Player actions
As a player, there are several actions you can make, including moving, using items and tagging other players.

• **Move:** a simple move to a neighboring cell. This is only allowed if the cell is a floor cell and is unoccupied. Moving into a wall cell results in no movement and moving into an occupied cell results in tagging the other player if he/she is from the other team and if you are currently on your team's area or the player has a flag, or being tagged if in the opposite team's area. If the layer is an allied player, the move is rejected.
•**Tag:** if you currently reside in your team's half of the map, attempting to move into another player's cell results in a tag if that player is of the opposite team. If you are in the opposite team's half of the map, this would result in you being tagged unless that player is holding your flag, in which case you would tag them. Moving into an allied player's cell has no result.
• **Using Jackhammer:** based on the direction attribute of the player and if facing a wall, pressing the action key of the jackhammer (tbd) results in the demolition of that wall, permanently turning it into a floor cell. Using the jackhammer on a floor, home, jail or outer wall cell has no effect, as well as using it on another player. Succesfully using the **jackhammer** results in a decrement of the amount of uses left, until there are no more uses in which case the **jackhammer** is destroyed.
• **Drop Object:** while holding an object (**flag** or **jackhammer**), pressing the drop object key (tbd) results in dropping that object onto the currently occupied cell. A dropped **jackhammer** does not reset its usage counter, and if picked up by another player, will have the same number of uses remaining.

4. Associated Rules

• **Move:** attempting to move to an adjacent cell results in the following rules checks:
    • **Unoccupied:** if the cell is unoccupied floor cell, your player moves into it.
    • **Wall:** if the cell is a wall cell, nothing happens.
    • **Enemy player:** if the cell is occupied by an enemy player, three things could happen: 1) if on your team's half of the map, you will tag that player, effectively sending them to jail; 2) if on the opponent team's half of the map, you will be tagged by that player, sending you to the enemy team's jail; 3) if that player has the flag, you will tag them regardless of which side of the map you're on, causing your flag to return to it's spawn location and sending that player to your team's jail.
    • **Teammate:** if the cell is occupied by a teammate, nothing will happen.
    • **Object:** if the cell is occupied by an object, you will pick up that object unless you are currently holding an object already in which case you will just move into the cell with no added results.
• **Tag:** refer to enemy player under the move action.
• **Jackhammer:** moving into a cell with a **jackhammer** object will result in picking up the **jackhammer**. When holding the **jackhammer** a player can press the associated key (tbd) to destroy a wall in the adjacent cell in the direction the player is facing. If there is no wall or an outer indestructible wall in the facing direction, nothing happens.
• **Drop object:** if a player is holding an object and the drop object key (tbd) is pressed, if there are no objects on the cell the player is occupying, the held object will be dropped and associated with that cell. Objects can only be dropped in floor cells, not jail or home cells.

5. Win conditions

In order for a team to win the game, that team's flag must be in it's original spawn position and they must carry the opposing team's flag into their home base. Additionally, all of that team's members must be inside the home area.

6. Semantics and Game Rules

• **Flags:** a player may pick up the opposing team's flag by moving into the cell containing it, as long as they are not already holding another object. Walking into a cll containing your own team's flag will return it to its original spawn point as long as it isn't already there. The flag will be coded as a struct with its current location, the ID of the player carrying it (if one is) a Boolean representing whether it is being carried or not, a boolean representing if it is at its original spawn point and a boolean representing if it is in the opposing team's base.
• **Guarding:** there will be certain rules against flag and jail guarding, which will prevent players of a team to be adjacent to a flag as long as it is in it's spawn point and preventing them from being adjacent to their own jail.
• **Jails:** jailed players can only move within the jail's bound as long as their status is "jailed". If a whole team is jailed, the opposing team wins. Players will be liberated if a teammate successfully walks into the jail and tags them.

Architecture

1. Server/Client

The server will maintain all of the game state data and logic. When a player makes a valid action, RPC requests and Event requests will be sent to the server, which will handle the requests in order of arrival, update the game state, and broadcast the new state to all clients.

The client will store the game state in itself so that it may be re-printed if requested and contain a minimal amount of logic that follows the following criteria: the client will contain action logic (how to respond to a specific action being taken by the player, e.g. moving, using objects, etc) and check for validity of the action. If a valid action is attempted, the client will not carry out the action but rather send a request to the server to carry out the action. However, if the action attempted is invalid, the client's action logic code will know this and instead of sending a request to the server that will result in an invalid action, it will simply reject the action, saving the server time from checking a useless action.