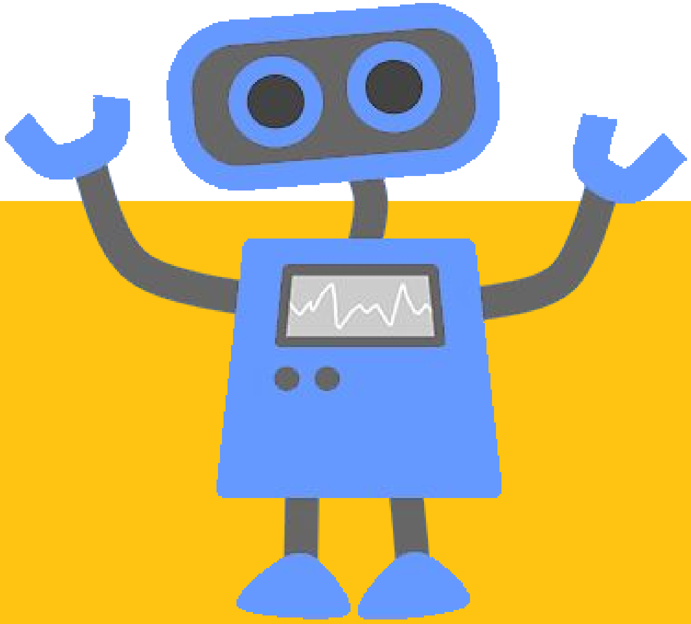
Madsweepers

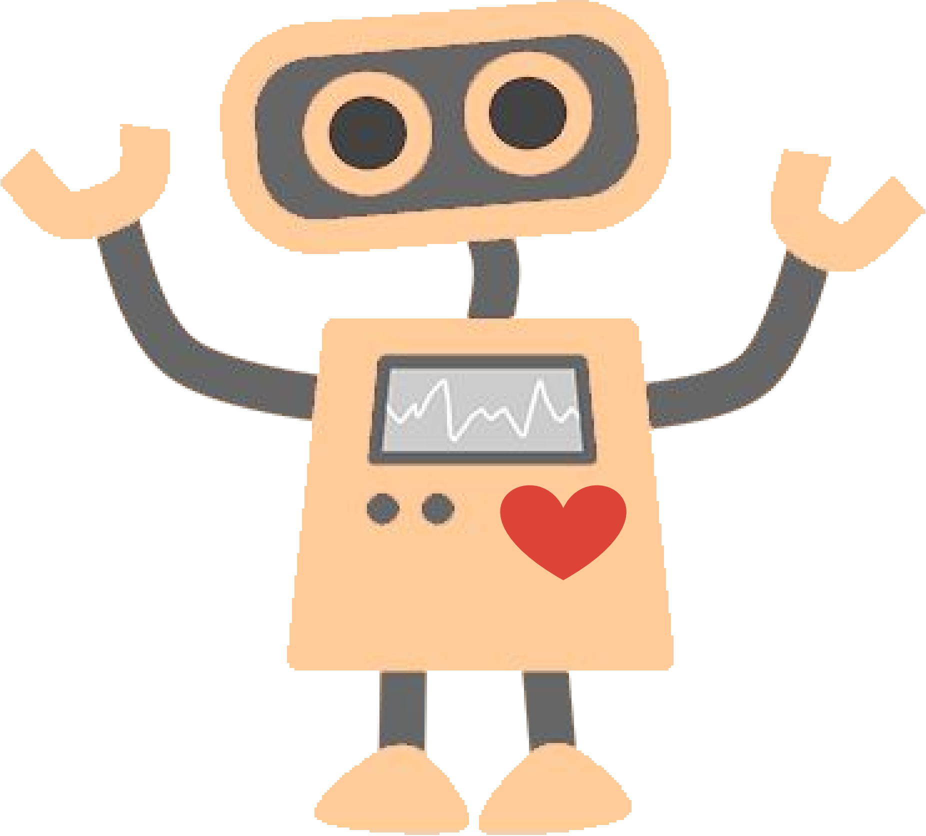
Multi-player minesweeper madness



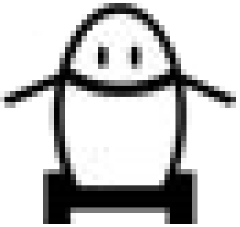
Bruce, Tim, MJ, David

Contents

1. How to play
2. Demo
3. Game Algorithm
4. Tech Stack
5. Challenges
6. System Architecture



How to play ❤



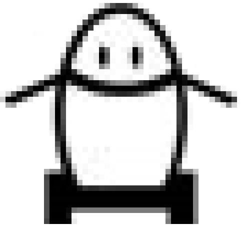
Click on a room to join

Click ‘ready’ when you’re ready to play

Arrow keys to move, ‘A’, ’S’, ’D’ to attack

‘F’ to flag mines, space to reveal tiles

The numbers on the board indicate how many mines are adjacent to that tile.



JOIN US FOR THE DEMO!

WWW.MADSWEEPERS.COM

GAME ALGORITHM

Game board is a matrix of objects containing all the information for each individual tile

Algorithm

1. Generates a board of objects with a default state of being empty
2. Determines the number of mines by multiplying rows by columns by the % of tiles that will be mines
3. Picks a random location to place a mine; if a mine is already there, tries again until successful
4. Goes through the matrix and calculates adjacent mines for each tile
5. Can generate a 1000x1000 board with 90% mines in half a second

Tech stack - Overview

FRONT END

react

Redux

TEST SUITE

mocha

chai

enzymeJS
sinonJS

BACK END

node

redis

socket.io

SYSTEM ARCHITECTURE

amazon
web services

docker

Tech stack - highlights

Real Time, Multi-Player

Challenge:

- Synchronize Changes for all players

Solution:

- **Web Sockets** to keep bidirectional channel between client and server
- **Redis DB** to provide real time change in leaderboard

Complex Game State

Challenge:

- Player locations and actions change fast and frequently

Solution:

- **Redux** to create deterministic state and view rendering, and one-way data flow architecture for our UI

Handling Traffic

Challenge:

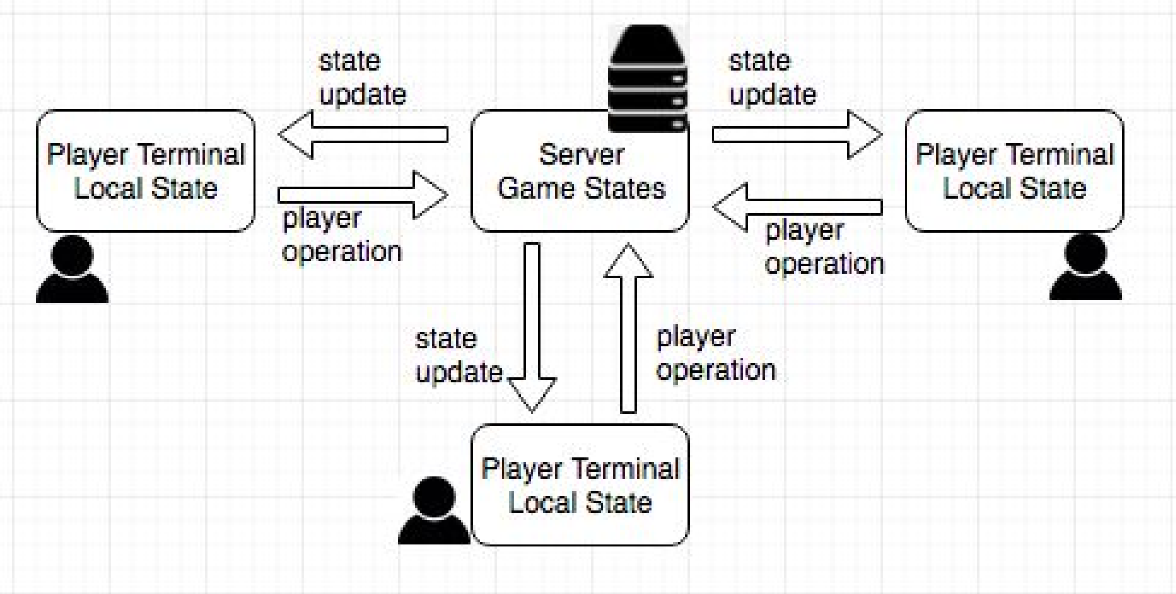
- User traffic is volatile (no users vs many players in a game)

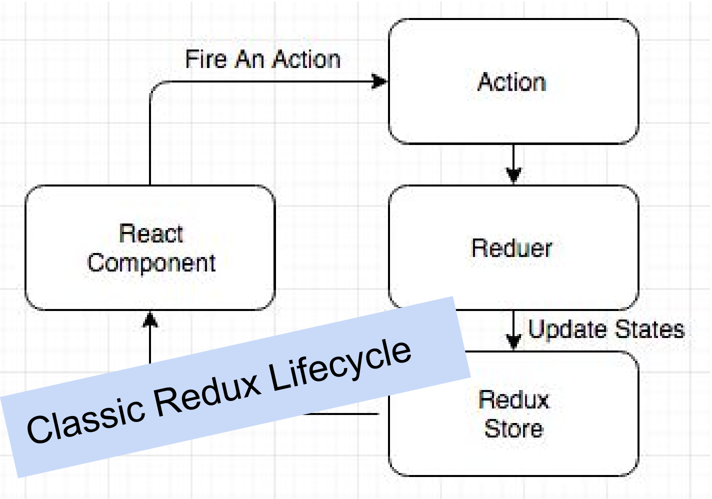
Solution:

- **AWS** with load balancer and auto scaling to handle different traffic loads efficiently

CHALLENGE!

Managing State & Socket Communication





Challenges

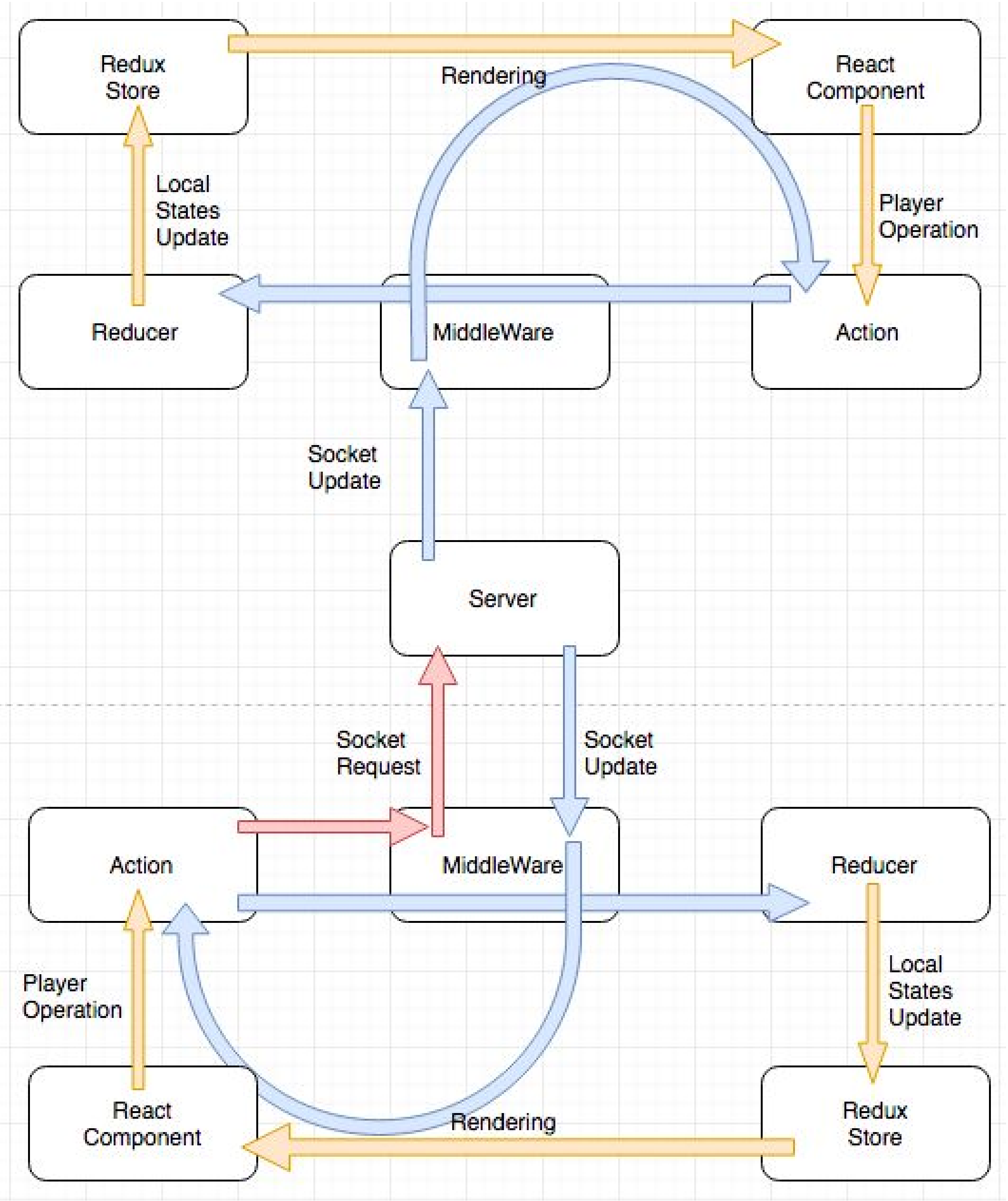
1. States changes can be triggered by different users
2. Frequent state changes
3. States aren’t stored locally, shipped to players through socket

Limitation

- Redux can only handle local states

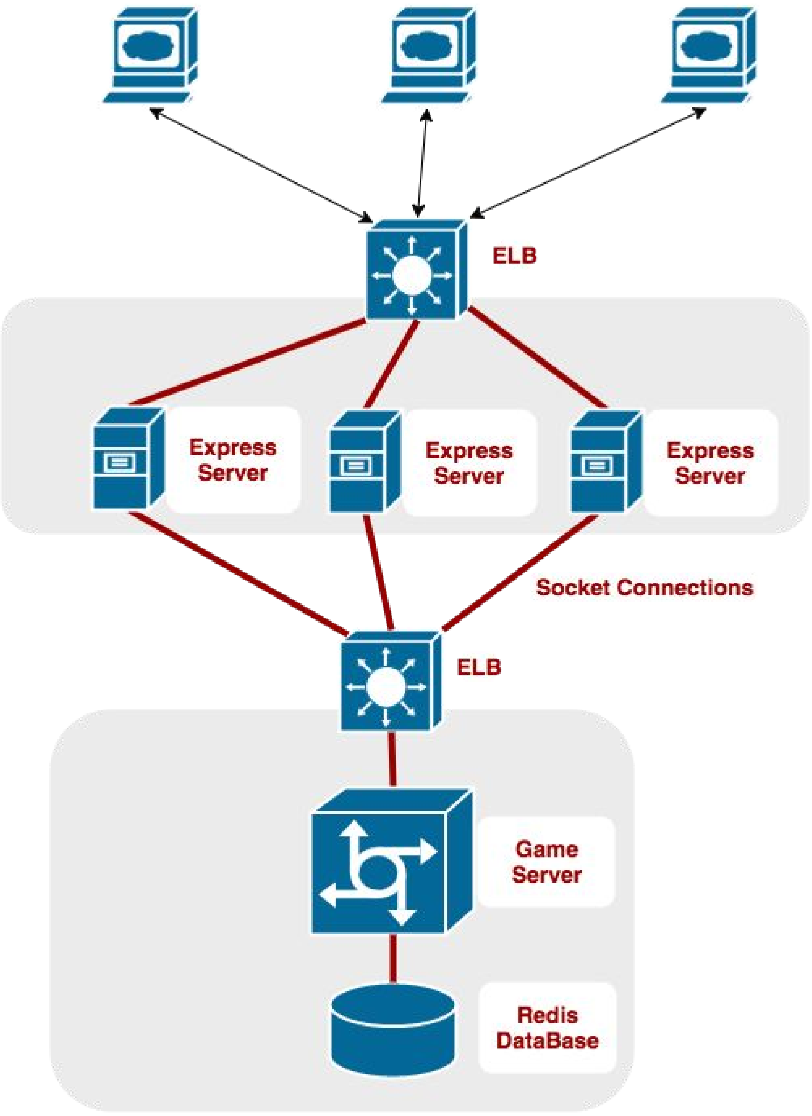
MiddleWare

- User fires an action
- **Middleware intercepts the action**
- Middleware sends out Socket Request
- **Calculation on Server**
- **Server broadcasts state updates through socket**
- **Middleware catches game update**
- **Middleware fires an action**
- Action passed to Reducer without being intercepted
- Reducer updates local states
- Redux states update forces Component rerendering

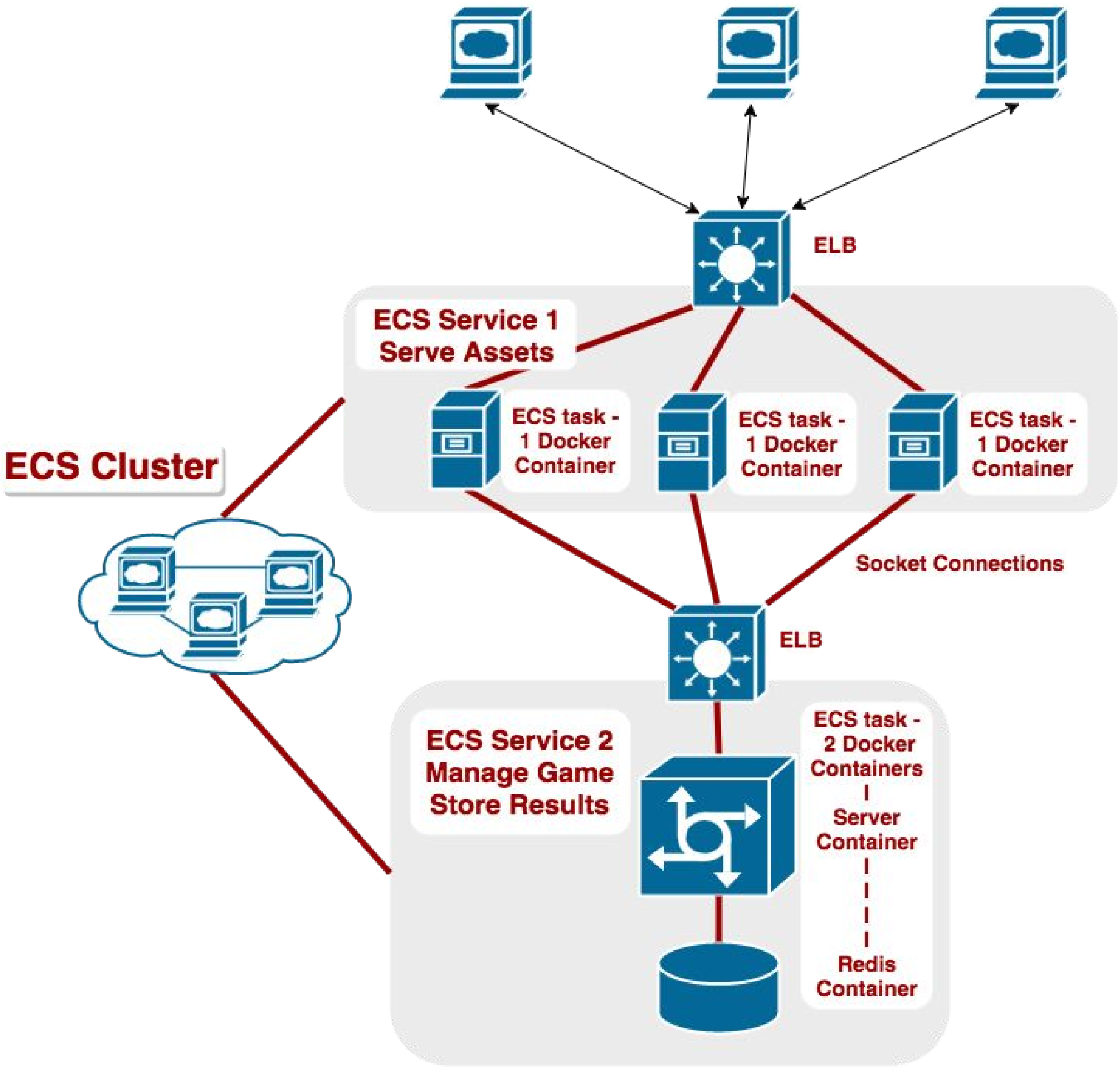


Redux Store
Rendering
React Component
Local States Update
Player Operation
Reducer
MiddleWare
Action
Socket Update
Server
Socket Request
Socket Update
Action
MiddleWare
Reducer
Player Operation
Local States Update
React Component
Rendering
Redux Store

System Architecture



System Architecture



Cluster

↓

Services (ELBs)

↓

Tasks (EC2 instances)

↓

Docker Containers