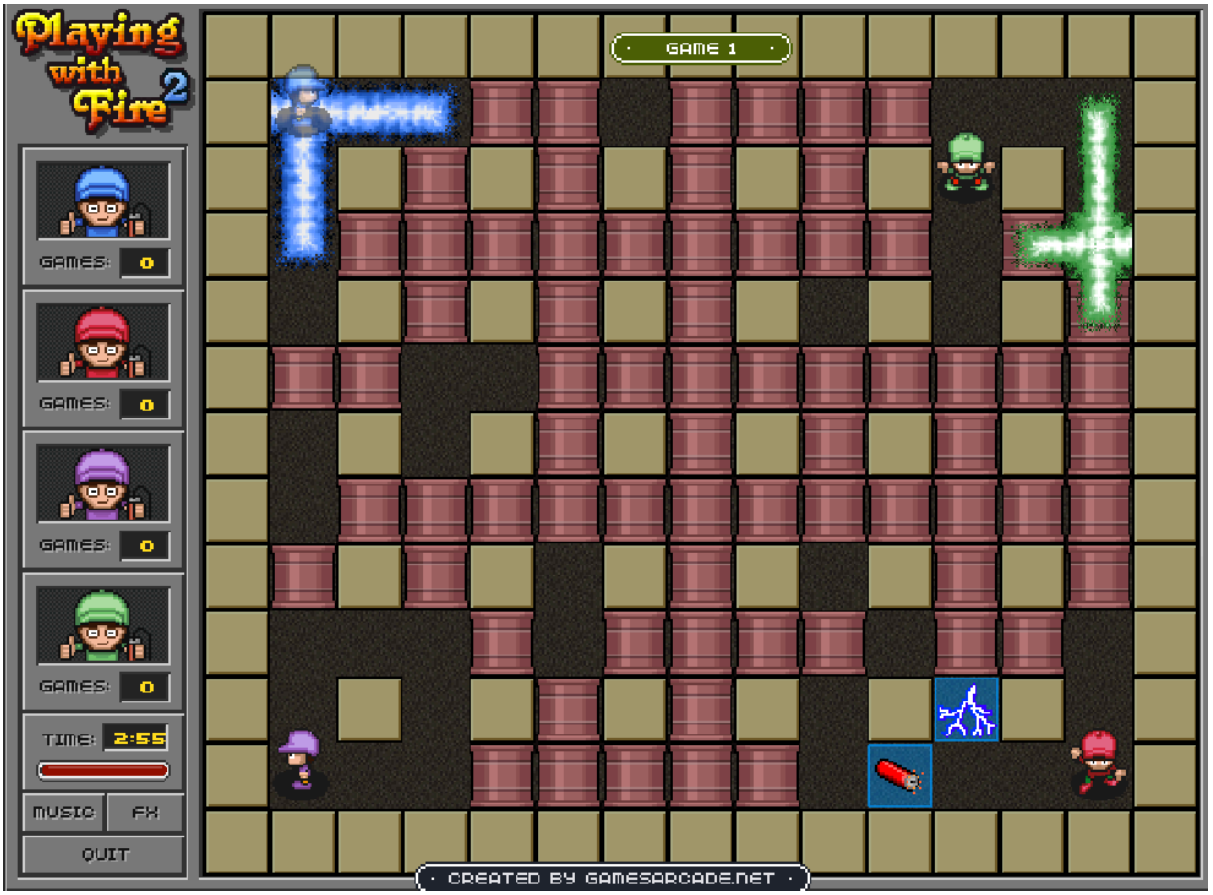
Playing with fire 2



Dolev Eisenberg

Roi Alus

System Architecture Overview:

This project implements a distributed version of *Playing with Fire 2* on a 14x14 map where the game runs across five physical machines, with the following architecture:

- Four Game-Nodes (GN1-GN4):
  Each game node is responsible for a quarter of the game map and the player using that machine (allowing for up to 4 players). It directly reads local keyboard input and runs that player's FSM. It also manages all processes within its quarter: map blocks, bombs, fire and any bots or items within its quarter. Each machine that hosts a GN also hosts a Local Graphical Node (LGN) - receiving information regarding the visual state of the entire map to display.
- One Control Node (CN): (fail-safe machine)
  The CN is responsible for game-wide coordination:
  - Receives game state updates from all game nodes.
  - Generate the complete game graphics, broadcasting it to all LGNs
  - Monitors all machines for failures.
  - Upon node failure, selects a replacement node and transfers control & latest game-state data to it. When the original node returns, transfers the game-state and control back to it.
  - Gathers statistics and general gamestate - # of blocks destroyed, game-time, power-ups etc.

Erlang Process Design:

- Map Tiles:
  One process per tile (wall, floor, etc.), ~200 tile processes on a standard 14x14 map.
- Dynamic Objects:
  - Each bomb is a process (handling countdown and explosion logic).
  - Each explosion/fire spawns processes for affected tiles.
  - Each item or power-up is its own process (before being picked-up). Possible power-ups include - extended explosion range, movable bombs, move-speed bonuses etc.

    - Effects that pass onto different nodes are being communicated between the nodes
- Player FSMs:
  Each GN runs one player FSM (human or bot) based on its own keyboard input. If no player is present, the node spawns a bot FSM instead. If a node running a bot disconnects from the network, the bot is passed to another node in the meantime.

Control Node Functionality:

The CN doesn't handle player input. Instead, it serves as the central observer and -controller - this node is the only node that cannot be disconnected (backbone of the system):

- Display:
  Receives periodic state updates from all GNs and renders the full game view.
- Failure Detection:
  monitors all game nodes. On a crash or disconnect, CN is notified.
- Recovery and Takeover:
    - Chooses one of the remaining GNs to take over the missing quarter and player.
    - Transmits the last known state of that quadrant and player FSM to the new host node.
    - The new GN handles the processes and player FSM

Tiles:

Each tile in the game is implemented as a separate *gen_server* process to support modular and scalable behavior. Since tiles maintain their own state, such as type (wall, floor, barrel), whether they are destructible, or if they contain an item encapsulating this logic in a dedicated process allows them to respond independently to events like explosions or power-up spawning. This design enables parallel updates across the map without centralized coordination. Each tile can interact with other entities in a clean, message-driven manner.

gen_server:

- Map tiles:
  Relatively static entities (floor, wall, barrel) with simple behavior, their responsibilities include: storing type, interacting with fire/explosions.
- Power ups:
  Passive until picked up. Their goal is changing player stats.
- Local Graphical Nodes:
  receives updates about the visual game-state from the control node.

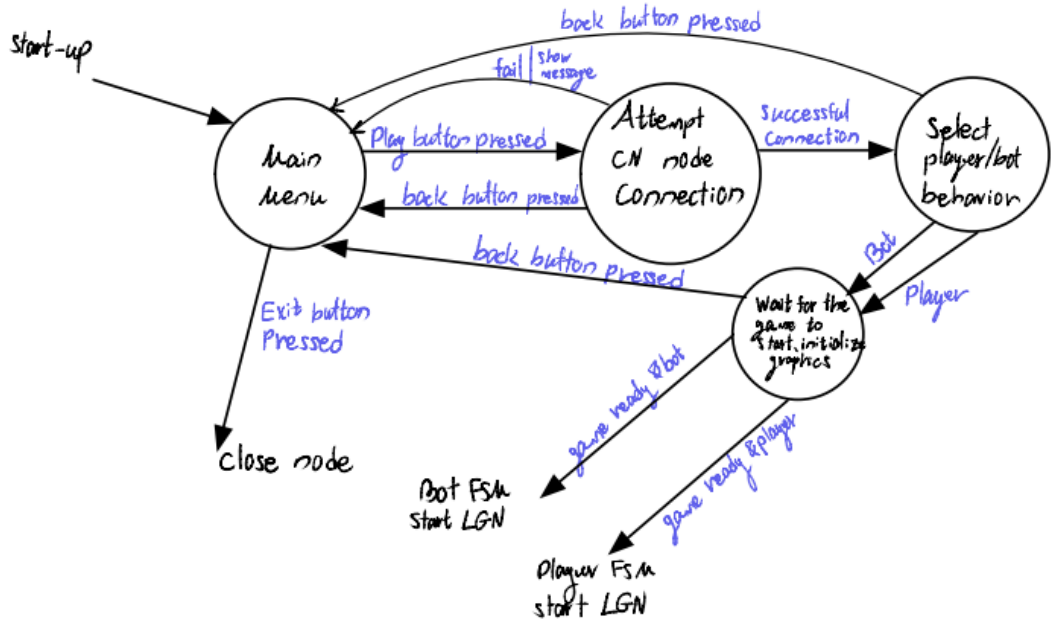
gen_statem:

- Player FSMs (Human or Bot):
  State based behavior (Idle, Move, Place bomb, death). Each player carries with it (using ETS) his power-ups, current lives, statistics etc.
- Bombs:
  State based, transition through distinct states (placed, countdown, explode), can be moved.
- Control Node:
  Manages failure detection and recovery in a sequence of modes like: monitoring, handover, recovery.

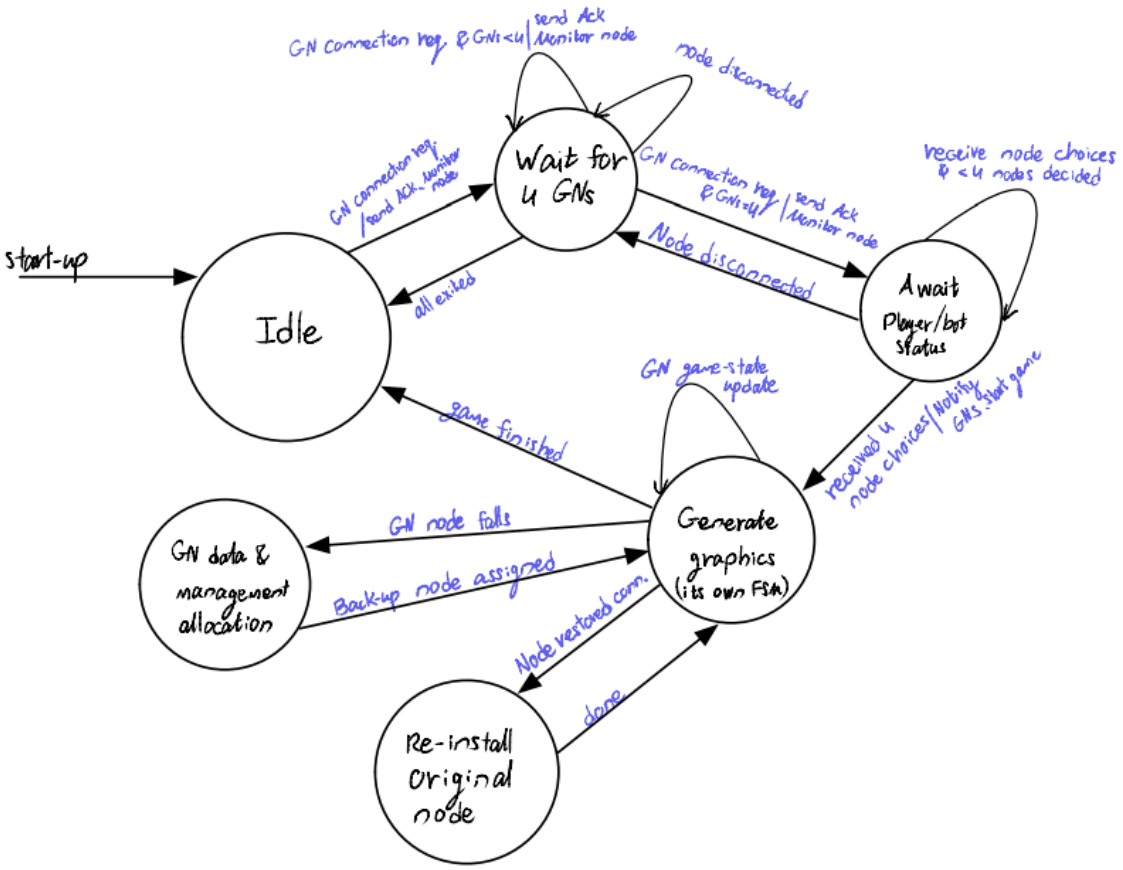
Non-User Interaction and Inter-Node Synchronization:

Each Game Node, even in the absence of direct user input from a local player, continues to operate and receive graphical updates at fixed intervals (every 50ms). Additionally, GNs are capable of receiving and processing events from other nodes in the system, such as: a bomb placement that affects their local quarter, damage to a character controlled by the node, collection of power-ups within the node's quarter (by any player in the game), destruction of tiles managed by processes within that node. For visual synchronization, the graphics are distributed by a single source (the CN) every fixed amount of time. This ensures the nodes are synchronized. On top of that, a GN can operate in "Bot Mode," in which it runs a player using a predefined FSM, which in that case no user input is inserted into that game node.

Player Node Main Menu FSM:



CN Overall Flow FSM:



User Interaction FSM: