Battle City

Multiplayer online battle arena game

COP 5618 – Dr. Sanders

Group 18: Junran Xie, Jingyu Luo, Yihao Wu

Gator Engineering
University of Florida
Department of Computer & Information Science & Engineering

Introduction

- This project produces a game similar to a classic video game, Battle City. Most elements, including pictures of tank, missile, wall and water, come from the original game. The difference is that Battle City is a local game which allows one or two player to control a tank to move in 4 directions, fire missiles, and fight with enemy tanks controlled by AI to protect the base, chapter by chapter, while this game is a multi-player online game. Each player controls a tank, fights with each other and wins the game by becoming the last one standing

Key Features I

Eliminate data race:

- The communication between client and server uses a double direction communication design. Game client uses separate socket server to accept messages and uses a socket client to send player operation messages to the game server, and the game server will create a socket client to send battle field message to game client's socket server. All data flows in a certain direction in one channel in order to eliminate data race.

Volatile variable:

- We use shared variables to communicate between multi-threads inside server or client. In these situations, we only need to guarantee that these variables change atomically and other threads are aware of the changes immediately. Hence the keyword volatile is sufficient.

Copy-on-write:

- When sending updated battle field, we follow the principle of copy-on-write. Server will make a copy of updated battle field and pass the reference to every socket client, to avoid real battle field is changed by other threads by accident, also to update the whole battle field in a consistent way.

Finer grained synchronization – lock splitting:

- We define different locks for different objects in BattleField, so that when battle field is accessed by multi-threads in different stages, the whole battle field won't be locked, and other threads may not be blocked by each other as long as their operations don't require the same lock.

Key Features II

Safe publication:

- To make sure battle field and tanks are fully created before being accessed by other thread, we use safe publication. All battlefields and tanks are created in static methods, which call private constructors inside.

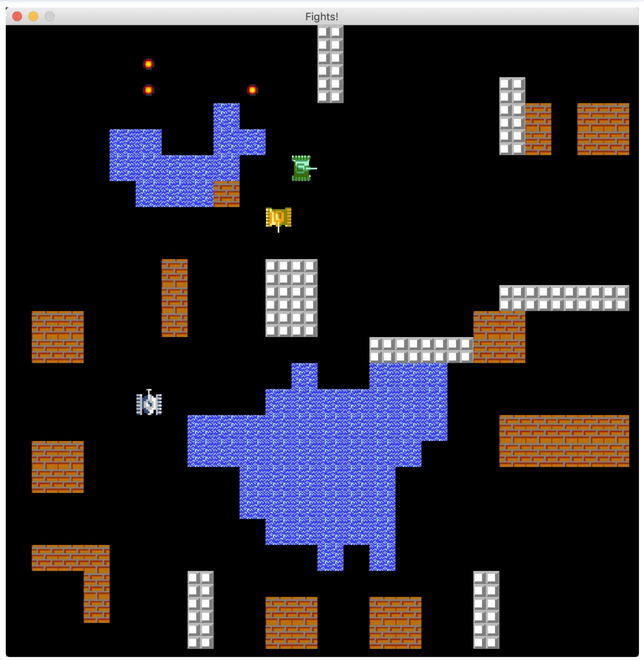
Safe cancellation:

- The loop conditions are controlled by a signal variable from battle field. As soon as one game is over, every thread will be aware of that and exit safely.

Strategy Selection

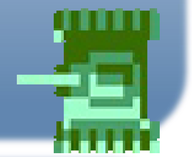
TCP or UDP:

- Generally, TCP is not necessary for online games because a frame's loss won't conduct vital failure while delay is more important for the experience of players. Hence UDP is more suitable for online games. However, there is still more work to do using UDP. For example, some crucial packets like handshake or result must be guaranteed to receive, and order of packets also needs to be kept. In this game, we only run it in LAN so that we can assume there won't be many packets loss, thus efficiency of TCP is sufficient.



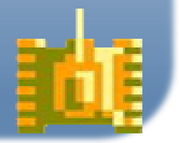
System Requirement

- Tested on Windows 10 and MACOS Mojave 10.14.4.
- Need at least two computers to make the game enjoyable.
- All clients and server must be in the same LAN.
- Firewall needs to be shut down.
- One client and one server can be run on the same computer.
- Support up to 5 players to play in one battle field. Maximum number of alive battle fields depends on hardware information.



Play Rules

- Up to 5 players can join a battle field.
- Each player control one tank and has individual start position.
- Tanks can move up, down, left, and right as long as there isn't any wall or water right in front of the tank.
- Tanks can fire missiles to the direction they face.
- Tanks can only move and fire with limited times in one second.
- Normal wall can be destroyed by missiles. Steel wall can not.
- Tank will be destroyed if hit by a missile.
- The last one alive is the winner.



Key Components

User Interface:

- Establish connection to server and initialize player interface.
- Listen keyboard operations of player.
- Display battle field to player.
- Report the winner.

Communication:

- Client send messages to server indicating player's operation, and receive messages from server to get updated battle field.
- Server receive messages from client to get player's operation, and send messages to Client indicating updated battle field.

Server:

- Initialize new battle field.
- Add tanks.
- Update battle field according to players' operations.

Post Project Thinking

- Are there better ways to start and stop the server? (Currently stop the server by control-C. There could be another main thread with a user interface that is responsible for start and stop server.
- Is it possible to implement the core logic of server using less lock or lock smaller scope?