
Taller de Programación

Cátedra Veiga

# Trabajo Práctico Final

Documentación Técnica

Carlos Matias Sagastume
110530

Martín Alejandro Estrada Saavedra
109236

Manuel Herrera Esteban
109883

# 1. Lenguaje usado

Para este proyecto se uso el lenguaje C++ (C++20) con el estándar POSIX 2008.

# 2. Arquitectura del servidor

Estructura del manejo de partidas: El server crea un hilo Acceptor y este se encarga de crear un hilo ClientHandler por cada cliente. Una vez elegida la partida, el ClientHandler crea un hilo Sender.

Para manejar múltiples partidas se utiliza un objeto de clase `Lobby` que tiene un `unique_ptr` de cada partida. La clase `Match` contiene toda la información de una partida.

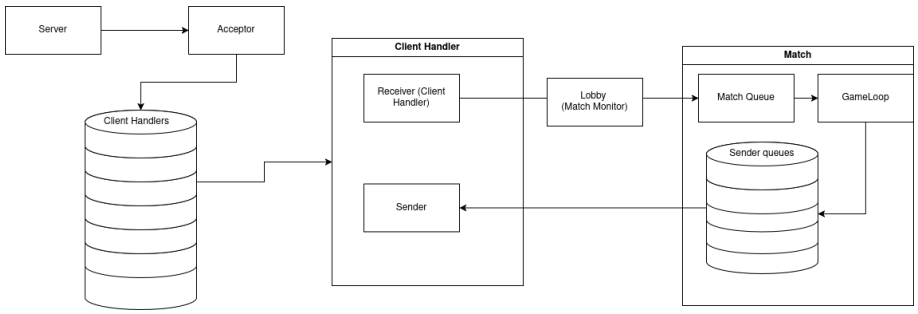


Figura 1: Diagrama del servidor

Estructura de una partida: `Match` es el encargado de leer y procesar las acciones de los jugadores. Para hacer esto utiliza diversas clases: `GameManager`, `GameClock` y `Map`. Luego de actualizar, obtiene el estado actual del juego (Snapshot) y lo envía a las `Sender Queues`. Para que el Sender envíe al jugador el snapshot.

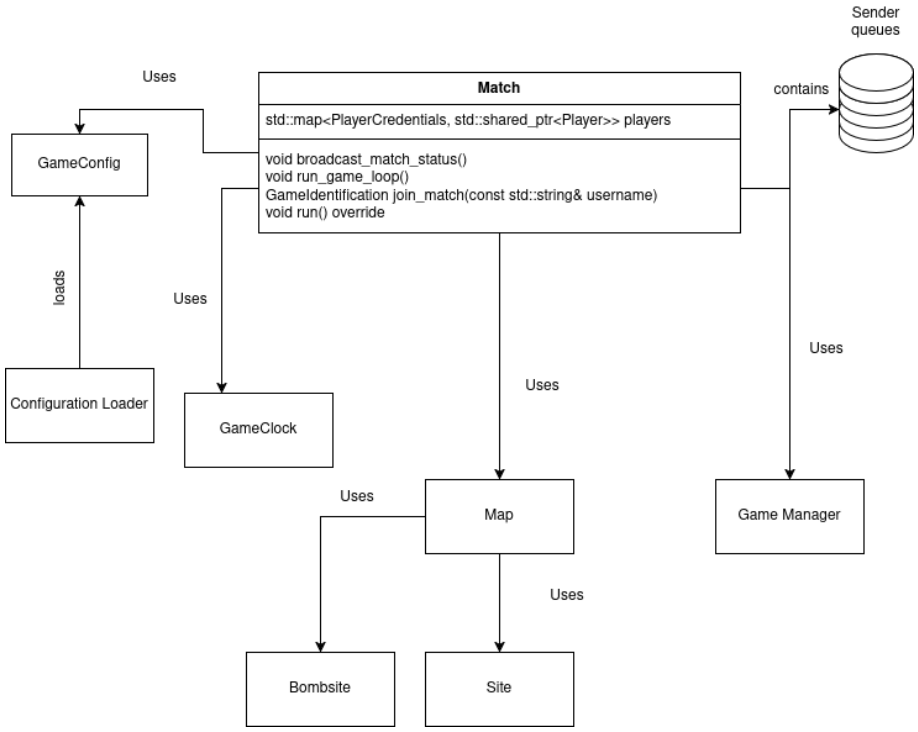


Figura 2: Diagrama de una partida

# 3. Arquitectura del cliente

Estructura del cliente: Una vez elegida la partida se crea un `ChatClient` y se ejecuta el método run. Este `ChatClient` a su vez lanza 2 hilos: `InputServerHandler` que es el encargado de recibir los mensajes de la partida desde el servidor y `InputHandler` que lee los inputs del cliente y los envía al servidor. Ambos hilos utilizan al `Protocol` que a su vez utiliza a la clase `Socket`. Ademas de esto, el ChatClient tiene un loop en el cual en cada iteración chequea si el servidor le envió una actualización y en caso de que si, lo renderiza. Este loop es un constant rate loop.

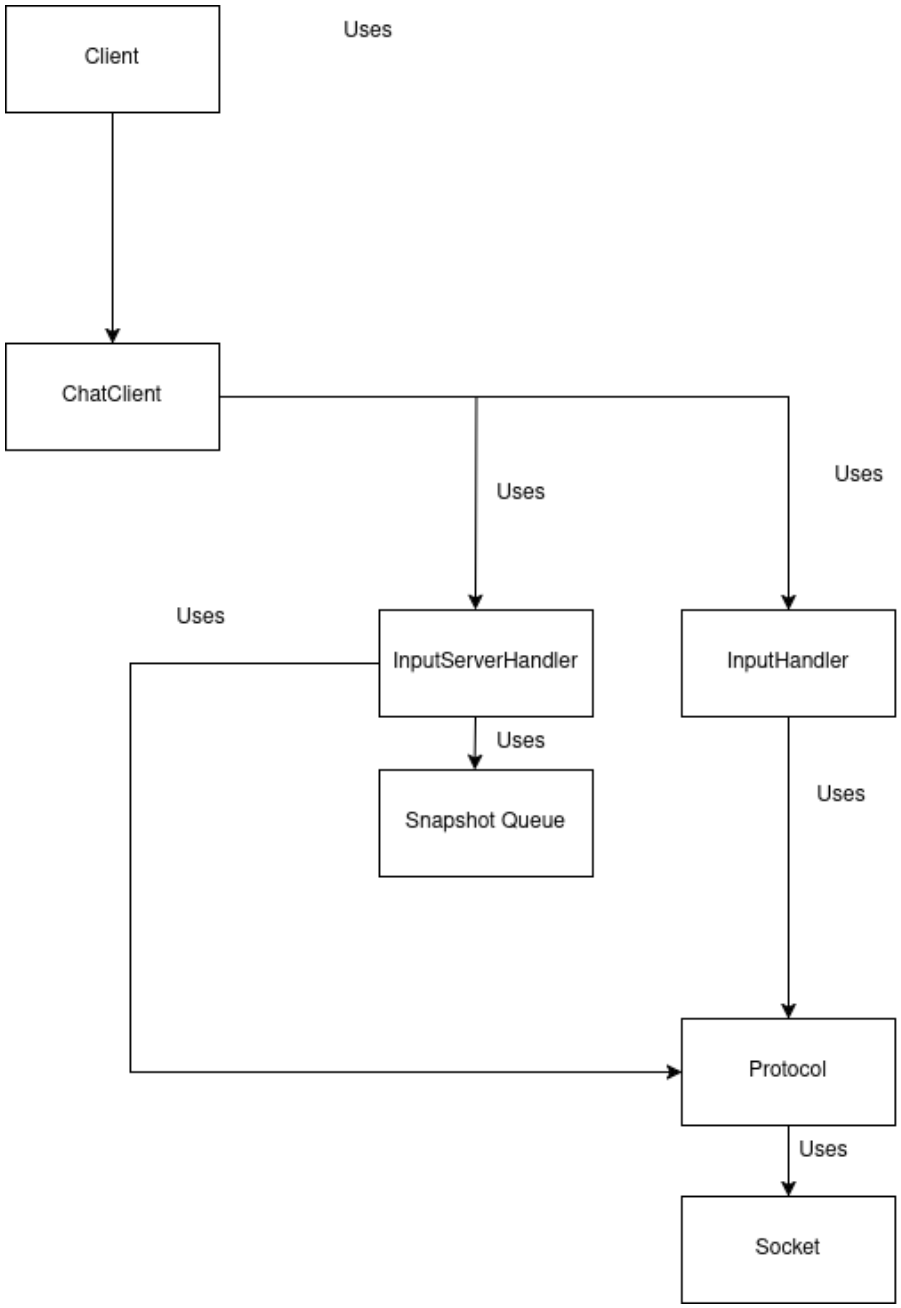


Figura 3: Diagrama del cliente

# 4. Comunicacion cliente-servidor

Ambos lados usan la misma clase `Protocol`, la cual se encarga de enviar y recibir mensajes a través de un Socket. Los mensajes enviados y recibidos son subclases de `Message` (superclase virtual). Algunos tipos de mensajes relevantes son: `PlayerAction` que es usado por el cliente para enviar las acciones al servidor, `GameStateUpdate` que es el mensaje usado para enviar las snapshots de la partida y `GameConfigInfo` que es enviado por el servidor al inicio de una partida conteniendo la configuración del juego.
Tipo de Acciones:
MoveUp
MoveDown
MoveLeft
MoveRight
Shoot
Reload
EquipWeapon
SetKnife
SetPrimaryWeapon
SetSecondaryWeapon
SetBomb
PlantBomb
DefuseBomb
Idle
BuyingWeapon
BuyingAmmo
Campos del GameStateUpdate:
bool game_started
bool game_ended
uint8_t round
float round_time
bool round_started
bool round_ended
bool bomb_planted
uint16_t bomb_x
uint16_t bomb_y
float bomb_timer
Team round_winner
Team game_winner
std::list<PlayerInfo>players
std::list<DroppedWeapon>dropped_weapons
Campos de PlayerInfo:
std::string user_name
uint16_t pos_x
uint16_t pos_y
uint16_t health
Status status
uint16_t money
uint16_t kills
uint16_t deaths
Action action
uint16_t pos_shoot_x
uint16_t pos_shoot_y
std::string skin
Weapon primary_weapon
uint16_t primary_weapon_ammo
Weapon secondary_weapon

uint16_t secondary_weapon_ammo
Weapon knife
Weapon bomb
Weapon active_weapon
uint16_t active_weapon_ammo
Campos de GameConfigInfo: int32_t player_health
int32_t number_of_rounds
int32_t starting_money
int32_t ct_amount
int32_t tt_amount
int32_t ammo_price
GunConfigInfo knife_config
GunConfigInfo glock_config
GunConfigInfo ak_config
GunConfigInfo awp_config
GunConfigInfo m3_config
int32_t defuse_time
int32_t time_to_plant
int32_t bomb_dmg
int32_t round_winner_money
int32_t round_loser_money
int32_t buy_time
int32_t bomb_time
int32_t after_round_time
int32_t money_per_kill
int32_t tiles_per_movement
int32_t game_rate
MapConfigInfo map_config
int32_t cone_angle
int32_t opacity
Campos de GunConfigInfo:
int32_t max_ammo
int32_t starting_reserve_ammo
int32_t min_dmg
int32_t max_dmg
int32_t gun_price
int32_t bullets_per_burst
int32_t shoot_cooldown
int32_t range
float angle
Campos de MapConfigInfo:
std::string map_name
int32_t map_width
int32_t map_height
std::vector<StructureInfo>structures
BombSiteInfo bombsite
SiteInfo ct_site
SiteInfo tt_site
Campos de BombSiteInfo: int32_t bomb_site_height
int32_t bomb_site_width
int32_t x
int32_t y
Campos de SiteInfo: int32_t x
int32_t y
int32_t site_width
int32_t site_height

`std::vector<std::pair<int32_t, int32_t>> spawns`

# 5. Librerias utilizadas

Librerías estándar utilizadas relevantes:
- random: para la generación de números aleatorios en la lógica de disparo y manejo de la bomba.
- cmath: usada para cálculos de distancia entre entidades y trayectoria de proyectiles.
- algorithm: utilización de diversos algoritmos.
- stdexcept: para lanzar excepciones.
- memory: para manejo de memoria, principalmente para el uso de smart pointers.
- chrono: en el servidor se usó para medir el tiempo por iteración en el game loop.
- mutex: utilizada para controlar el acceso concurrente a las partidas.
- iostream: utilizada para imprimir mensajes de error.
- arpa/inet.h: utilizada en el protocolo para el manejo de endianess
- thread: utilizada para el uso de hilos (en algunos casos, en otros se utilizó la biblioteca provista por la cátedra).

Librerías externas utilizadas relevantes:
- yaml-cpp: utilizada por el servidor para parsear el archivo yaml de configuración.
- Socket provisto por la cátedra: Utilizado por el cliente y el servidor para comunicarse.
- Thread y Queue provisto por la cátedra: Utilizados para el manejo de hilos y el traspaso de información entre éstos.
- Qt y SDL: utilizadas por el cliente para diseñar la interfaz, los gráficos y los sonidos.