Project Specification – 2DShooter

Names: Saketh Neeli, Prakhar Rathore, Dale Liu
Period 5

Overall Description

Our game is a top-down 2D multiplayer shooter game using Java using the LibGDX framework for rendering and KryoNet for networking. Players control pixelated characters on a top down perspective, playing on a 2000×2000 pixel map in a high-speed combat game. Movement is handled via the keyboard (WASD), and aiming and shooting are directed using the mouse. Players can choose different weapons, with different stats such as damage, fire rate, spread, bullet speed, and reload time. Firing has an animated recoil and directional bullets with collision detection.

The game uses a client-server model where all clients connect to a central server via KryoNet. The server has real-time data exchange, including player movement, bullet synchronization, hit detection, and player health updates. Each player is going to be continuously updated about the positions and actions of others, so we can have real-time multiplayer gameplay. Bullets are synced across clients and 'removed' upon hitting walls or players, with collision managed using rectangular hitboxes and circles for bullets and players.

The game map consists of a background PNG for visuals and a collision layer which are made by a set of hardcoded rectangles using a Tiled collision map. The camera follows the player, keeping them centered on the screen. The map includes walls, crates, and other structures that players cannot walk through. Bullets break when hitting these obstacles.

Weapons are selected at spawn, and players will automatically respawn after a short delay when they die. Guns include cooldowns, reloading, and ammo counts. The project structure is modular, including classes like Player, Bullet, Gun, GameMap, and GameScreen.

Class/Interface Overview

Game Logic

- **GameScreen.java** – The primary game loop and rendering screen. Handles player input, updates

player and bullet positions, renders the map and players, and processes networking events.

- **Player.java** – Represents a local player in the game. Stores position, health, current gun, velocity, and rendering logic.

- **EnemyPlayer.java** – Represents remote players in multiplayer mode. Similar to Player but updated from network data.

- **GameMap.java** – Loads the background map and obstacles. Stores collision rectangles and handles their rendering. Integrates with map files and handles collision detection with the environments

Weapons

- **Gun.java** – Represents a weapon type with attributes like fire rate, damage, bullet speed, and spread.

- **Bullet.java** – Represents a bullet in the world. Stores its position, velocity, damage, lifetime, and handles movement updates

Networking

- **GameClient.java** – Manages the client-side KryoNet connection. Sends and receives player/bullet updates, connects to the server, and manages client state.

- **ClientListener.java** – Handles incoming messages on the client-side and updates local game state accordingly (e.g., bullet hits, enemy positions).

- **GameServer.java** – Manages server-side logic. Tracks all connected clients, their positions, and broadcasts updates to all clients.

- **ServerListener.java** – Listens for events from connected clients and updates the server's copy of the game state.

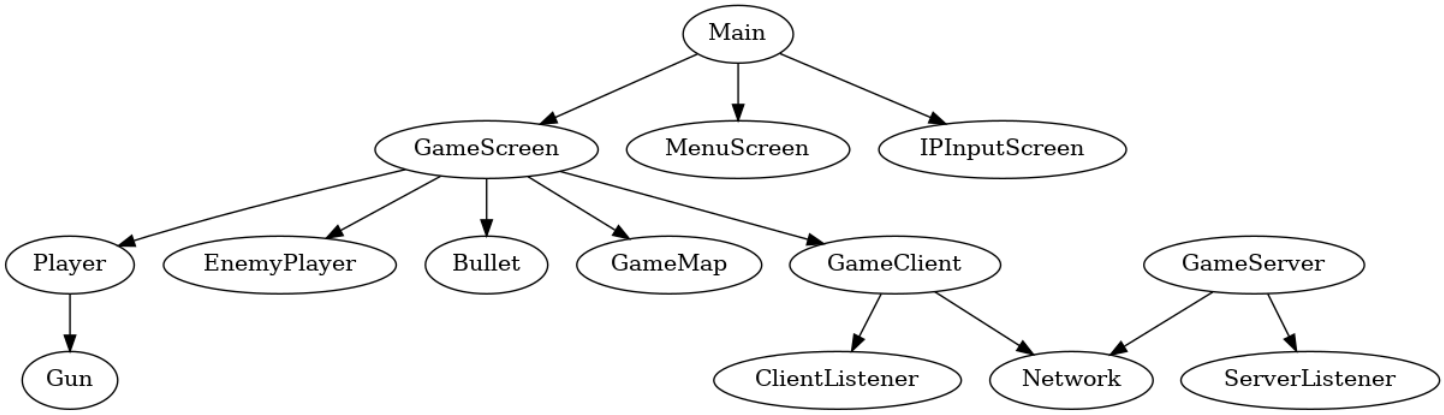
- **Network.java** – Registers all KryoNet classes and packet types used for communication between server and clients.

Screens

- **MenuScreen.java** – Displays the game's main menu and handles navigation to other screens like game start or IP input.

- **IPInputScreen.java** – Allows the player to enter a server IP address to join a multiplayer game.

Rough Class Diagram



Structural Design

The following data structures will be used.

Description	**Structure**
Obstacles	Array<Rectangle>
Other players (multiplayer)	HashMap<Integer, PlayerData>
Bullets	ArrayList<Bullet>

Data structure rationale

We chose to use an Array for obstacles since there are a set amount of obstacles in the map and whenever the player moves we have to check for any obstacles which means we have to iterate through all the obstacles. This is very efficient for our purposes and there are not that many obstacles so it did matter too much which data structure we used for this. Using the Intersector class from LibGDX we can quickly check for player collisions.

For handling multiple players in multiplayer we used a HashMap with player ids as keys and PlayerData as values. Using this data structure ensures O(1) lookup times when we need data from different players which happens every game loop which is a lot of times which is why lookup times need to be very efficient. When people leave we also have to handle deletion which is efficient with HashMaps.

Bullets in the game are managed using an ArrayList which fits our use case specification. We continuously remove and add bullets from the ArrayList so we are creating and removing bullets when they are fired. Adding new Bullets is an O(1) operation. Using a fixed Array would not be useful as different weapons have different amounts of bullets.

High Level Major Class Specifications

GameScreen

- **Attributes**
  - Main game instance
  - OrthographicCamera camera
  - Viewport viewport
  - ShapeRenderer shapeRenderer
  - SpriteBatch batch
  - Player player
  - GameMap map
  - ArrayList bullets
  - Vector2 velocity
  - float[] accel
  - Vector3 mousePosition
  - Vector2 aimDirection
  - boolean multiplayer
  - String serverAddress
  - GameClient client
  - InputAdapter input
- **Methods**
  - show()
  - render(float dt)
  - handleInput()
  - updateBullets(float dt)
  - checkBulletCollisions()
  - resize(int w, int h)
  - pause(), resume(), hide(), dispose()

Player

- **Attributes**
  - Circle hitbox
  - Vector2 velocity
  - Texture texture
  - float health
  - float speed, maxSpeed
  - boolean alive
  - long respawnTime
  - float rotationAngleDeg
  - Array guns
  - int currentGunIndex
  - boolean isFiring
- **Methods**
  - update(float dt, float w, float h, Array obstacles)
  - handleGunInput()

    - fireAt(float dirX, float dirY)
    - takeDamage(float dmg)
    - respawn(float w, float h, Map<Integer, ?> otherPlayers)
    - shouldRespawn()
    - render(SpriteBatch batch)
    - renderGunInfo(SpriteBatch batch, BitmapFont font, float x, float y)
    - set/get methods for velocity, health, rotation, speed, etc.
    - dispose()

GameMap

- **Attributes**
    - Texture background
    - Array obstacles
    - TiledMap map
- **Methods**
    - render(SpriteBatch batch)
    - getObstacles()
    - dispose()

Bullet

- **Attributes**
    - Vector2 position
    - Vector2 velocity
    - float radius, speed, damage
    - long creationTime, lifetime
    - int ownerId
    - boolean stopped
- **Methods**
    - update(float delta)
    - stop()
    - isOutOfBounds(float w, float h)
    - isExpired()
    - get/set methods for damage, lifetime, radius, position, velocity, ownerId, stopped

EnemyPlayer

- **Attributes**
    - Circle hitbox
    - static Texture texture
    - float rotationAngleDeg
    - boolean alive
    - float health
    - boolean textureInitialized
- **Methods**
    - initializeTexture()
    - update(float x, float y, float health, boolean alive)
    - render(SpriteBatch batch)

    - setRotationAngleDeg(float angle)
    - get methods for position, health, hitbox
    - disposeTexture()

Gun (Abstract)

- **Attributes**
    - String name
    - float damage, fireRate, reloadTime, bulletSpeed, spread
    - int magazineSize, currentAmmo
    - boolean isReloading
    - long lastShotTime, reloadStartTime
    - Color gunColor
    - float gunMaxLength, gunLength, gunThickness
- **Methods**
    - fire()
    - reload()
    - update()
    - gunRecoil()
    - get methods for attributes

GameClient

- **Attributes**
    - Client client
    - int clientId
    - Map<Integer, PlayerData> otherPlayers
- **Methods**
    - sendPlayerUpdate(...)
    - sendBulletShot(...)
    - sendPlayerHit(...)
    - updateOtherPlayer(...)
    - initializeEnemyTextures()
    - getOtherPlayers()
    - disposeAllEnemyPlayers()
    - close()
- **Inner Class: PlayerData**
    - int id
    - float x, y, health, rotation
    - boolean alive
    - Circle hitbox
    - EnemyPlayer enemyPlayer
    - update(...)

ClientListener

- **Attributes**
    - GameClient gameClient
    - BulletListener bulletListener

    - PlayerHitListener playerHitListener
- **Methods**
    - received(Connection, Object)
    - setBulletListener(...)
    - setPlayerHitListener(...)

GameServer

- **Attributes**
    - Server server
- **Methods**
    - start()
    - stop()

ServerListener

- **Attributes**
    - Server server
- **Methods**
    - received(Connection, Object)
    - connected(Connection)
    - disconnected(Connection)

Network

- **Constants**
    - int port = 54555
- **Inner Classes**
    - PlayerUpdate
    - BulletUpdate
    - PlayerHit
- **Methods**
    - register(Kryo)