BulletZone Report

Project: BulletZone

Team: Refactored (Dexter Richards, Taylor Quinn)

12/6/14

Table Of Contents

1. Introduction
2. Software Requirements
    1. Functional Requirements
3. Design Specifications
    1. Class Diagram (Figure 1)
    2. Sequence Diagrams (Figures 2-4)
    3. Design Patterns
    4. Database Tables
4. User Documentation
    1. Screenshots
5. Software Tests

Introduction

A game server exists that handles all of the server-side functions of a game called BulletZone, but there is no client-side user interface that allows players to interact with the server and play the game. The purpose of this project is to create that user interface using Android mobile devices to allow players play the BulletZone game. This in turn will allow us as programmers to apply object-oriented design principles to a concrete project.

Software Requirements

- Functional Requirements
  - Data to be sent to the server:
    - Player tank moves (backwards or forwards)
    - Player tank fires a bullet
    - Player tank turns (up, down, left or right)
  - Data to be received from the server:
    - A 2D array of integers representing the playing field and what's where
    - The health points of all the tanks
    - Where each entity is on the grid
    - Player tank's health
  - Each player can only control their own tank
  - There can only be two bullets from each tank on the screen at a time
  - Tanks cannot move into walls
  - Start Screen functionality:
    - Have a splash screen with an image and the game's title, BulletZone
    - Have a start button that will bring the user to a play screen
  - Play Screen functionality:
    - Swipes will send move and turn functions
      - If the tank is facing up and the user swipes up, the game will send a “move up” event to the server
      - If the tank is facing up and the user swipes left, the game will send a “turn left” event to the server
      - Etc...
    - Provide Join, Leave and Replay buttons
      - Join will join the server and register the player's tank ID
      - Leave will quit the game and send a “leave” event to the server
      - Replay button will only enable after the player has died and will let the user rewatch the events that lead to their death

Design Specifications

- Class Diagram: See Figure 1
- Sequence Diagrams

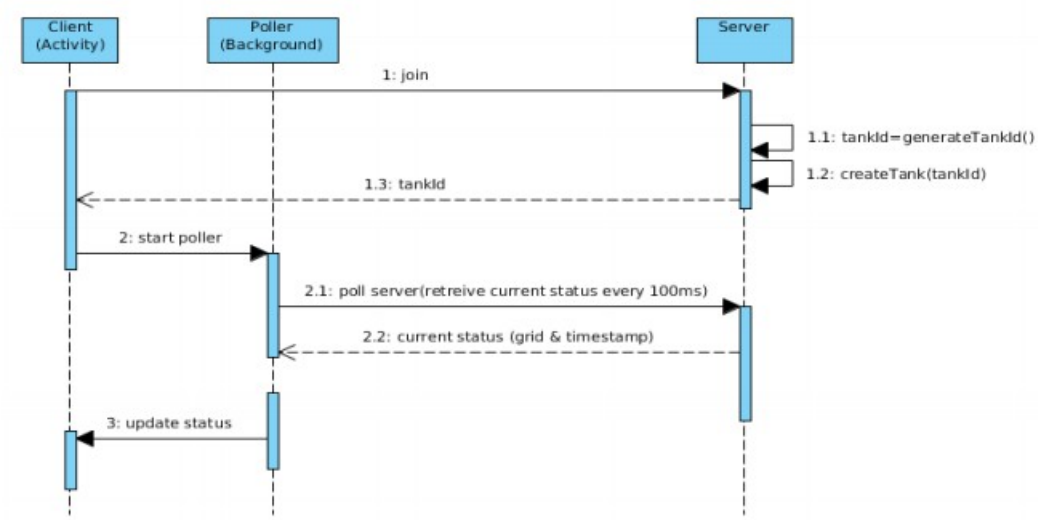

Figure 2: Joining the Server

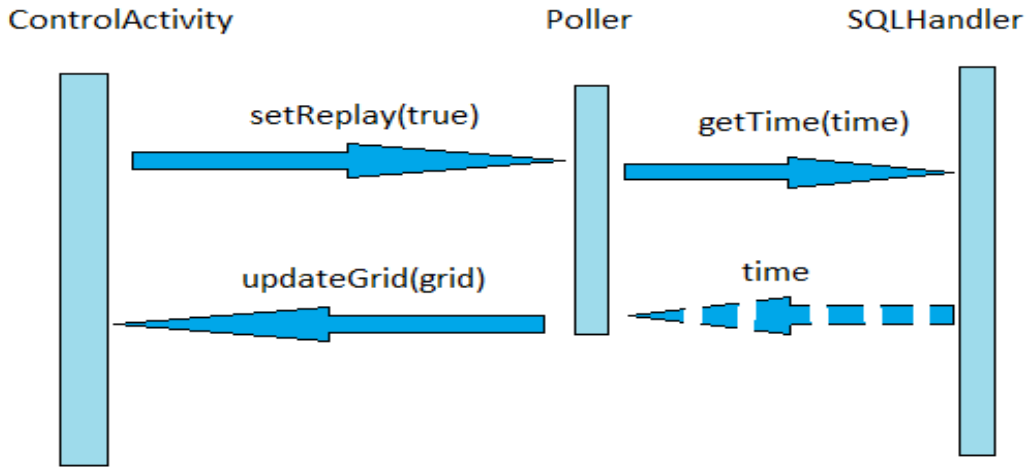

Figure 3: Replay

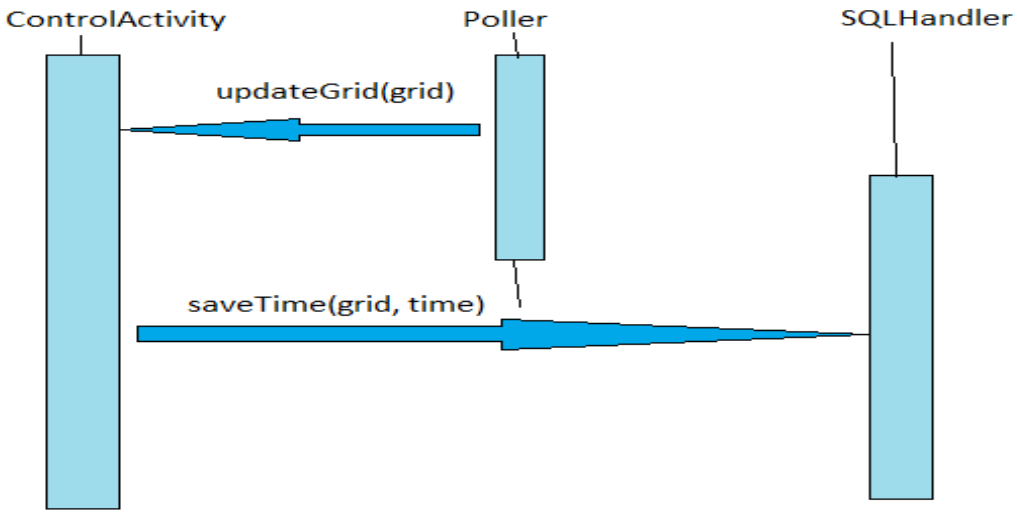

Figure 4: Save Time

- Figure 2 (Join):
  When ControlActivity sends the join event to the server, it also starts up the Poller, which will grab events from the server every 100 milliseconds. The server will send a response back to ControlActivity with the player tank's ID.
- Figure 3 (Replay):
  Within Poller, there's a loop that constantly polls the server for updates, and passes these events back to ControlActivity. However, if the inner variable “replay” is set to true, the Poller will instead grab frames from the SQL database and pass those to ControlActivity, instead. To do this, ControlActivity tells the Poller what the current time is and it rewinds through the last ten frames continuously until the user leaves or starts a new game.
- Figure 4 (Save Time):
  As described above, the Poller grabs grids from the server and updates ControlActivity with them. When this happens, ControlActivity passes these grids to the SQLHandler, which puts them into the database to later be fetched, should the replay event be called.

- Design Patterns Used:
  - Observer Pattern: the main activity receives events from the server using an observer pattern
  - Singleton Pattern: the event bus used in the observer pattern is fetched using a singleton pattern
  - Factory Pattern: the map entities are created using a factory pattern
- Database Tables:
  - Table “grids”
    - int id
      - Auto-incremented
      - Primary key
      - Not used
    - varchar(2000) str
      - String that holds the entire 2D grid at each time
      - Each value is comma-separated
    - int timestamp
      - The time of the saved grid event
  - Additional Notes:
    Originally, the design would take in the entity objects and translate them into individual values and then insert them into Tank, Bullet and Wall tables, but this made it run entirely too slow to be useful. Then, the design was modified to use a simplified table consisting of integer x and y coordinates, the integer timestamp and the string delivered by the 2D int grid. This, too, was too slow to be useful, which lead to the current implementation.

- Sounds:

The sounds were implemented in order to receive extra credit, serving as a buffer for other places that may have lacked. Joining the game, moving the tank, firing the tank, and dying each have sounds with the file extenstion ".wav" located in the "raw" folder.

User Documentation

After pressing the Start button on the main splash screen, the main game field will be displayed. Pressing the Join button will join the game and display the events of the game. To move your tank, swipe in the direction the tank is facing to move forward. Swiping in the opposite direction will move the tank backwards. To turn, move in one of the directions the tank is not facing. To fire, shake the device.

Screenshots:

Illustration 2: Starting Splash Screen

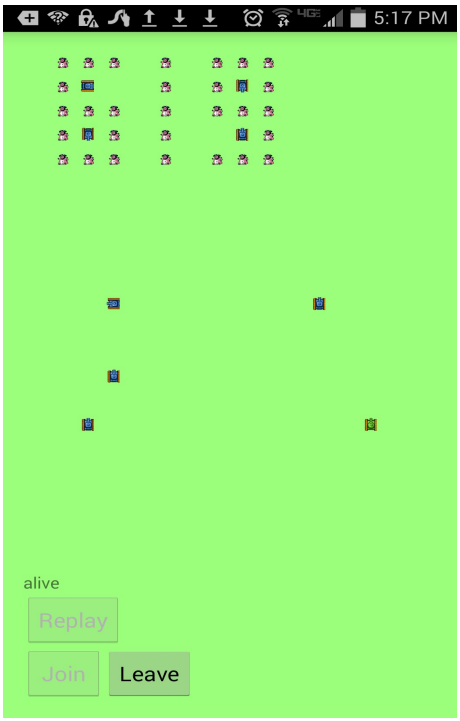

Illustration 1: Play Field

Software Test Plan:

There are currently 20 tests that check for each of the entities such as the tank, bullet, wall, as well as the UI entities such as the UItank, UIbullet, UIdestructible wall, and UI indestructible wall. The first set of tests go through and make sure that every parameter is accounted for by setting and getting the values. Then the tests try to make a UI entity out of the entity. The tests pass if the image for the UI matches the image that is expected. Another important test is "Zbullets", which checks the constraints of the game. This test shows the ability to iterate through the field and ask anything about the game including bullets, walls, tankid's, directions, healths, walls, deaths and more.