XTetris: Official Report

Mirco De Zorzi
<mircodezorzi@protonmail.com>

1 Team Composition

The program has been written on by a single person: Mirco De Zorzi (mat. 891275).

2 Project Structure

2.1 Toolkit

- pre-commit: runs `clang-format`, `clang-tidy`, `oclint`, `uncrustify`, `cppcheck` and `cpplint` on commit to guarantee code consistency;
- Makefile: controls the generation of object files and executables;
- ccache: object caching to speed up re-compilation;
- gcc: compiler.

2.2 Scaffolding Tools

To generate the project scaffold `mircodezorzi/secretary` was used. Running the following command will create a repository with all tools described in section 2.1.

```
$ secretary create libdz --template c-make
```

2.3 `libdz` Architecture

`libdz` uses a callback model for all event handling, from keyboard and mouse events, to button clicks. What follows is a minimal *working* example of the high level API that the library exposes:

```
void on_loop(engine_t* e) {
  // draw ...
}

void on_keyboard(engine_t* e, kb_ev_t ev) {
  // handle keyboard event ...
}

void on_mouse(engine_t *e, mouse_ev_t ev) {
  // handle mouse event ...
}

engine_t *e = engine_new(
  on_loop, on_mouse, on_keyboard);
engine_loop(e)
```

2.4 Multiplayer Implementation

Multiplayer orchestration is handleed through a centralized server written in Go, and ideally running on a publicly reachable endpoint, such as a EC2[1] instance. For demonstration porpuses only the game server is going to be run locally, along side the two clients.

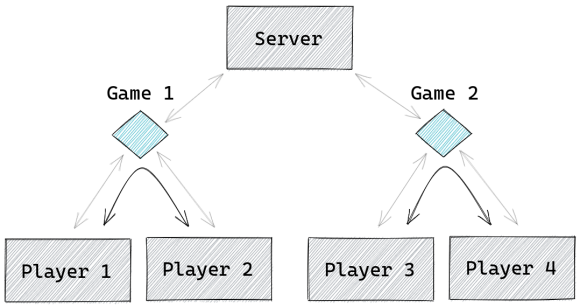

Figure 1: Client-Server Model. Black arrows represent data flow from the client's prespective, while the gray ones represent what actually happens. `Game N` represents the abstracted data structure that holds the games's state.

3 Unimplemented Features

- Both client-side and server-side authentication and session handling is somewhat brittle. In case one of the clients times-out, there is no way to recover a game session;
- The UI library (`libdz`) is still in a *work-in-progess* state. Most components require a close-to-complete rewrite, and the code has yet to be hardened, so security is not guaranteed. Some missing features from the library are:
  - non-blocking TCP client and server;

[1] https://aws.amazon.com/ec2/

```
type userConn struct {
        conn    net.Conn
        scanner *bufio.Scanner
        state   connState
        user    string
        field   []int
        room    string
}

type room struct {
        A *userConn
        B *userConn
}
```

Listing 1: Simplified data structures used by the server. Refer to `server/main.go` for more context.

  - scenes: state-machine management by the engine to automate navigation between them panels;
  - panels, textboxes, checkboxes...: layout automation for complex menus (refer to `mircodezorzi/sym` for an close implementation);
  - better bounding-box handling for all widgets: allow for a widget to have a `parent`, altering it's origin point in relation to the parent's one.;
  - pass `context_t*` to all callback functions instead of the parent object;
  - ...
- Better field state handling. Currently it is possible to send arbitrary information to the server, effectively making it possible to set one own's field to an arbitrary state. This would lead to trivially simple cheats.
- There are no options menu to change field size, key bindings or multiplayer settings.
- Player Vs. AI is missing.

Refer to the source and header files for more documentation related to unimplemented features.