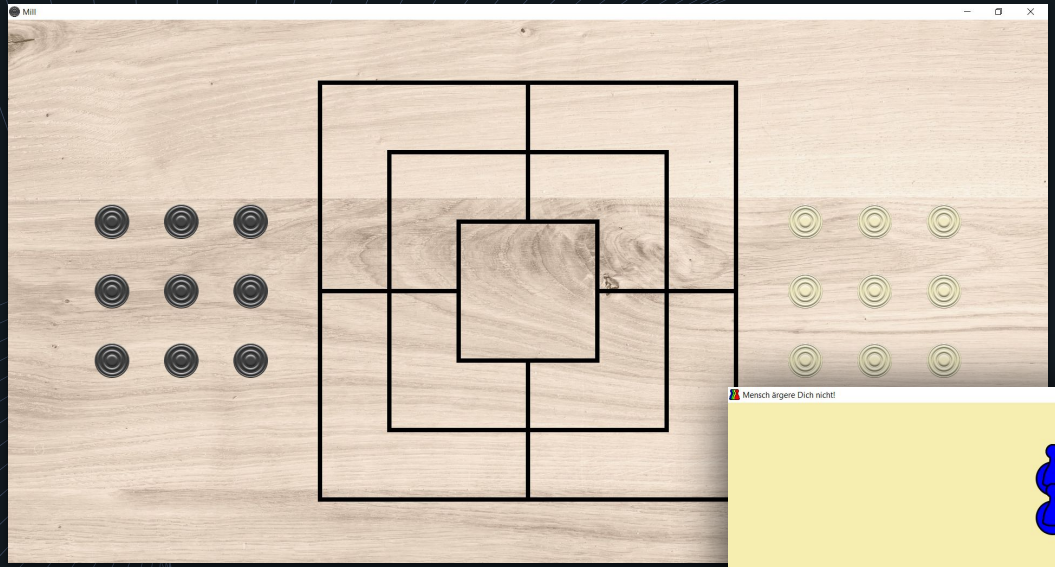
Board Game Framework

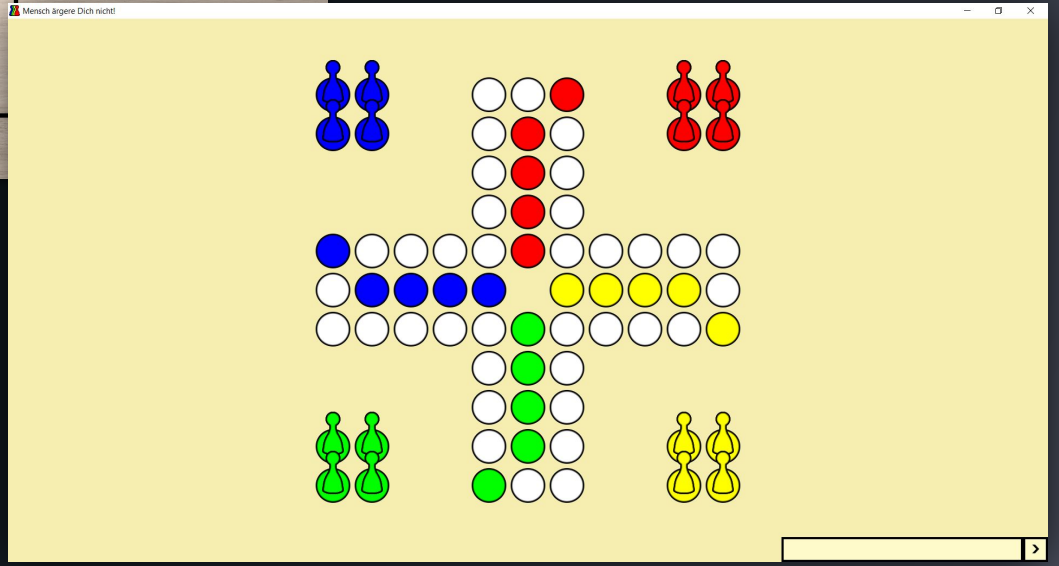
Eder Julia - Schwaiger Fabian - Thiele Luis

January 20th, 2022

Table of Contents

Introduction


Mill


Mensch ärgere Dich nicht!

Used Dependencies

- Java Development Kit 16
  - Java Sockets
- LWJGL 3 (“Lightweight Java Game Library”)
  - OpenGL
  - GLFW
  - JOML
- Gson
- Maven

https://www.lwjgl.org/customize

Game Loop

```
while(!isCloseRequested()) {
    currentMillis += System.currentTimeMillis() - lastMillis;
    lastMillis = System.currentTimeMillis();

    if(currentMillis >= MILLIS_PER_TICK) {
        update();
        currentMillis -= MILLIS_PER_TICK ;
    }

    render((double) currentMillis / MILLIS_PER_TICK) ;
}
```

Render vs. Update

Render Tick

- All ticks are render ticks
- Irregularly, untimed, unsteady
- Update visual representation (interpolation)
- A.k.a. “frame”, “FPS”

Update Tick

- Not all ticks are update ticks
- Regularly, timed, steady
- Update game logic, input listeners, received packets
- A.k.a. “tick”, “TPS”

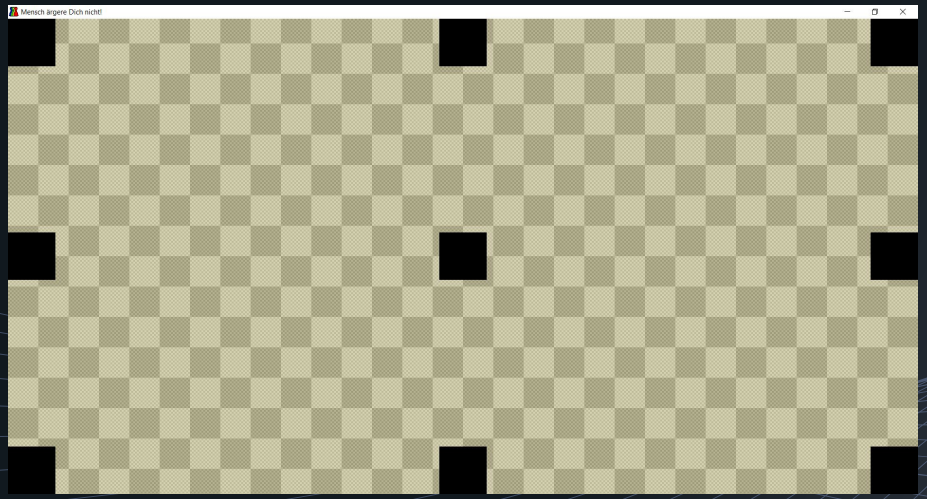
Engine

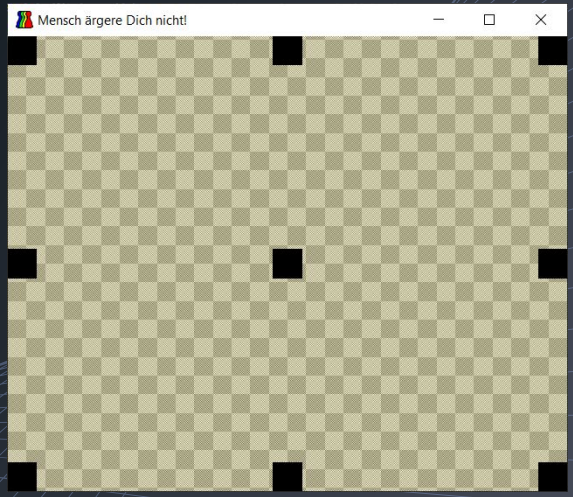
- 1 main game instance
- 1 engine instance
- 1 window instance
- ...

→ Singleton design pattern

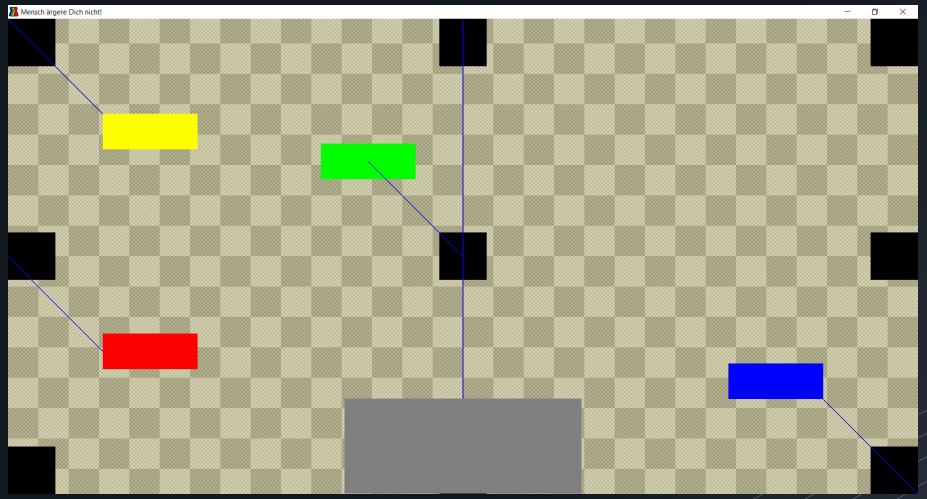
User Interface FW

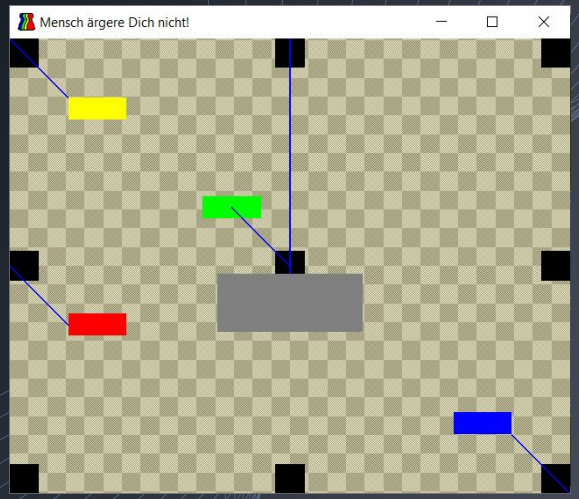
Anchor-Points





Anchor-Points





Widgets

- No-functionality widget (visuals only)
- Button widget
- Selectable widget (checkmark)
- Text input widget
- ...

- Ludo nodes, figures, dice animation...

Widgets

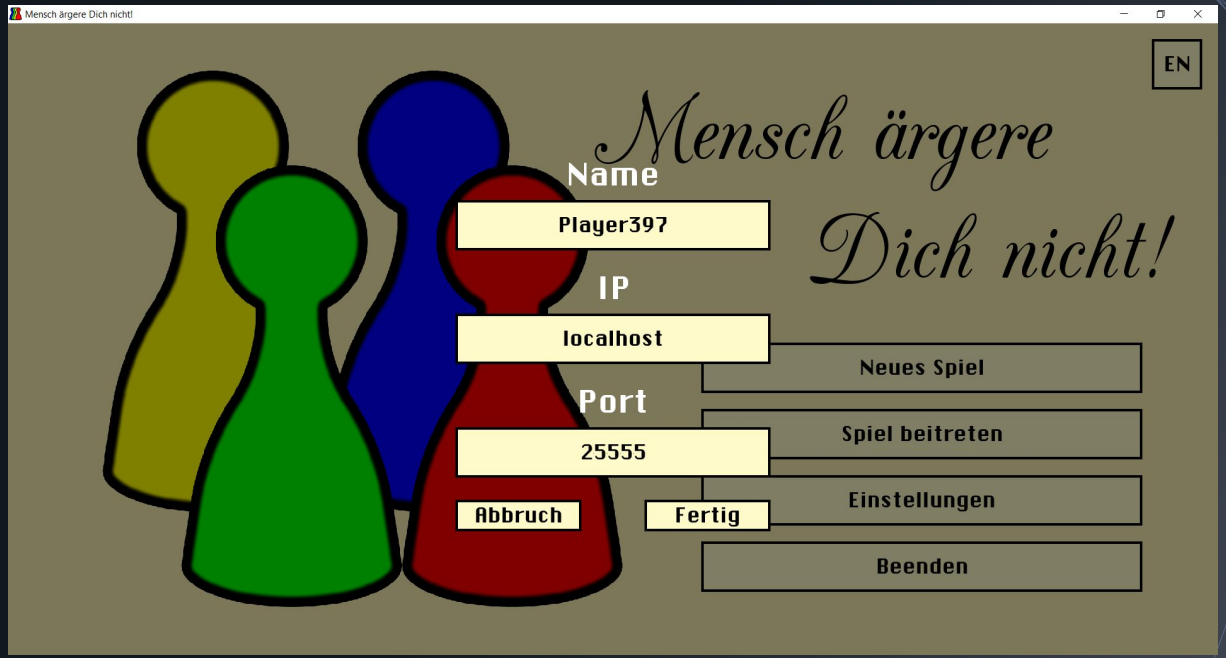
```
void update(int mouseX, int mouseY);
void render(float deltaTick, int mouseX, int mouseY);
void mouseButtonPressed(int mouseX, int mouseY, int mouseButton, int mods);
void mouseButtonReleased(int mouseX, int mouseY, int mouseButton, int mods);
void keyPressed(int key, int mods);
void keyRepeated(int key, int mods);
void keyReleased(int key, int mods);
void charTyped(char character);
```

Styles

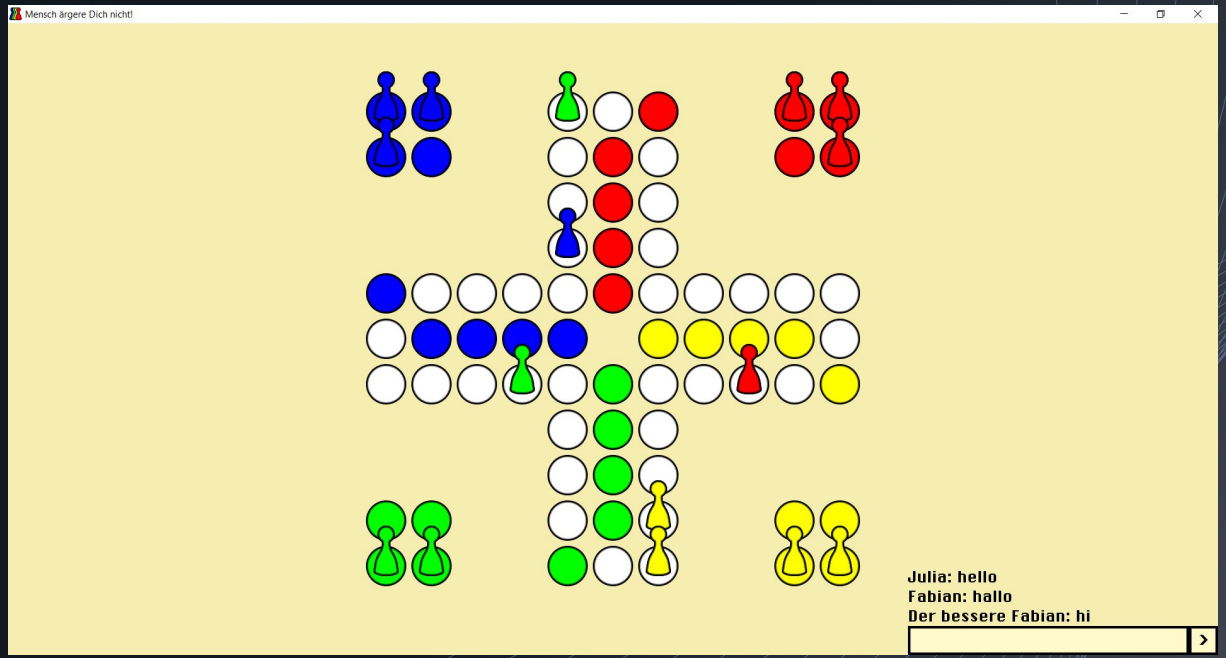
- Textures
- Text
- Colored quads and borders
- Mouse-hovering based styles
- ...
- Styles are stackable, customizable, mergeable...
- → Decorator design pattern

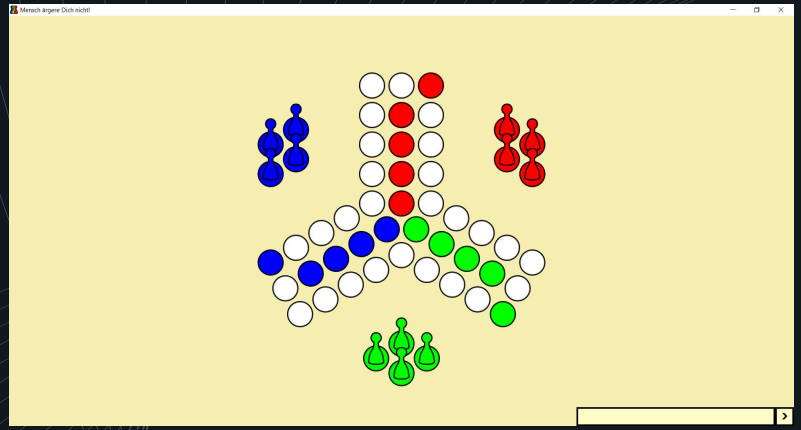
Example Screens

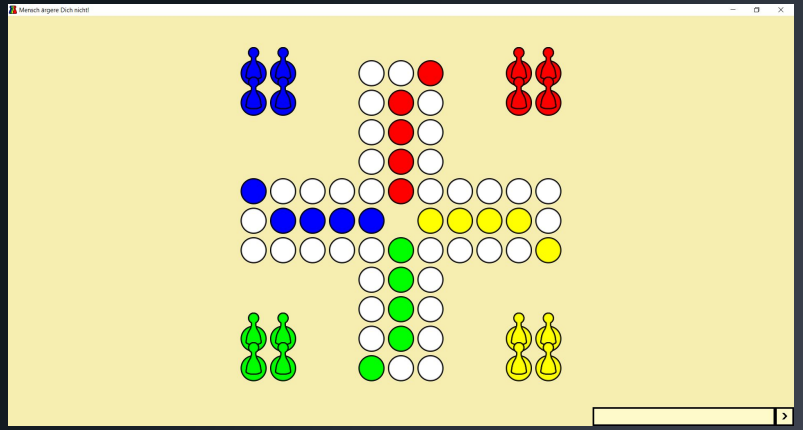
Example Screens

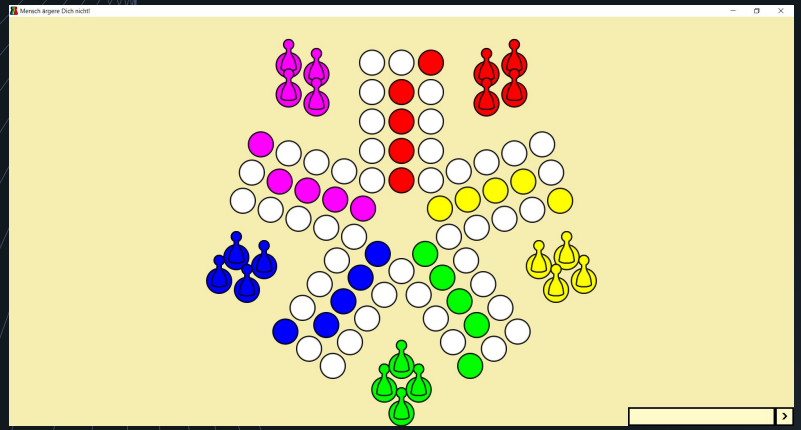


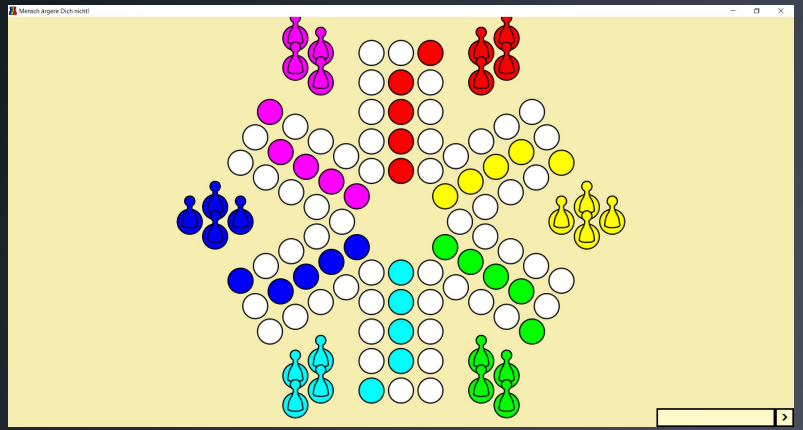
Example Screens




Mensch ärgere Dich nicht!

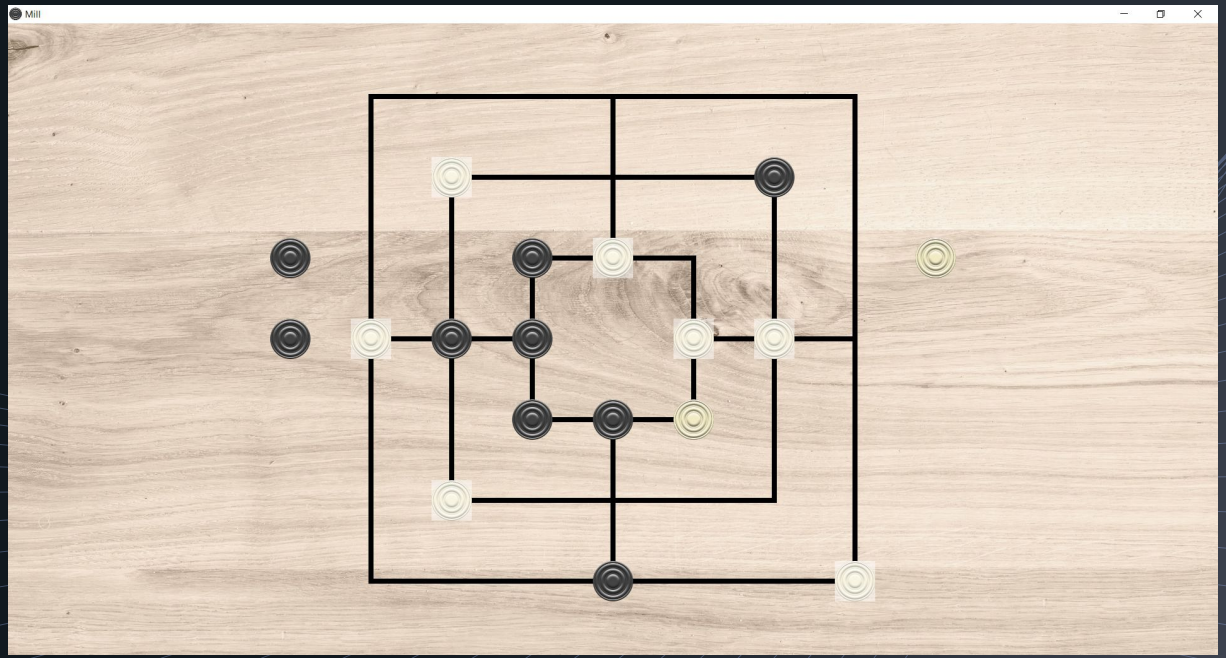

Mensch ärgere Dich nicht!


Mensch ärgere Dich nicht!

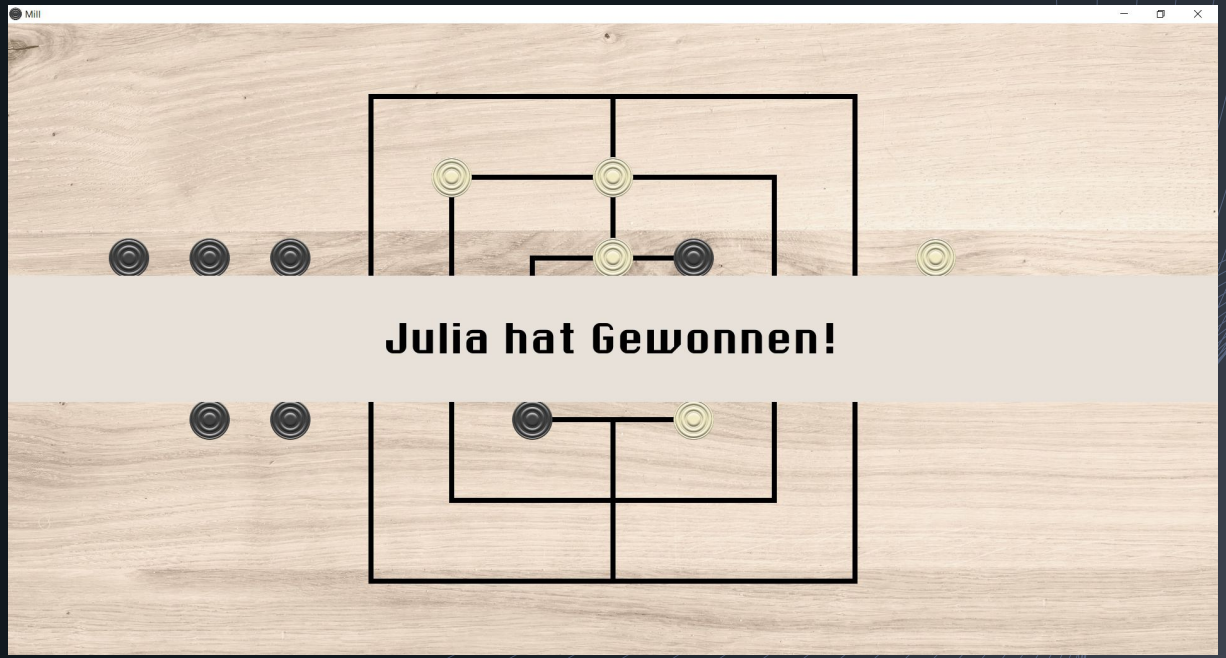

Mensch ärgere Dich nicht!

Example Screens

Example Screens



Example Screens



Board Game FW

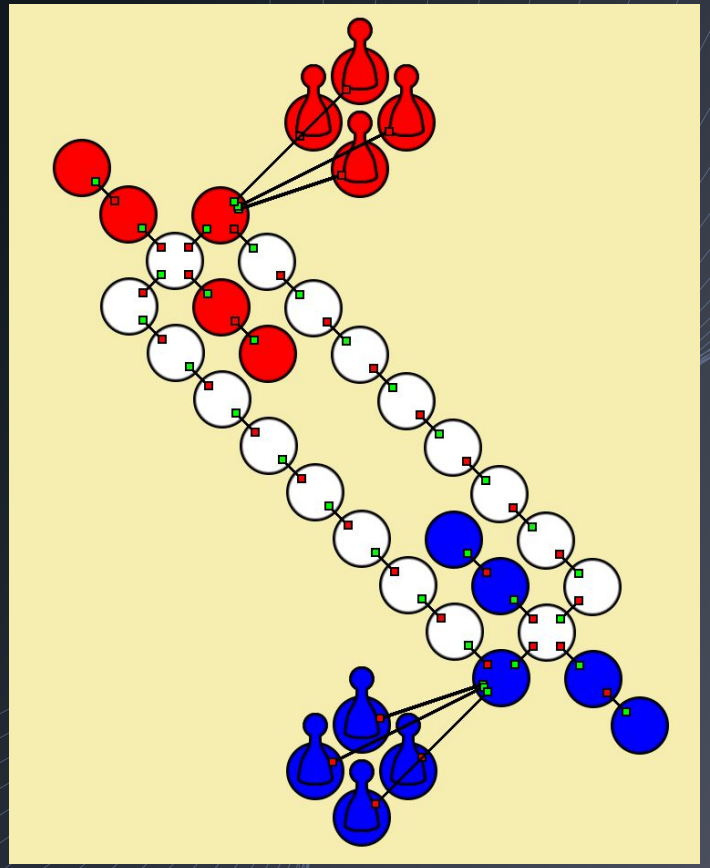
Structure

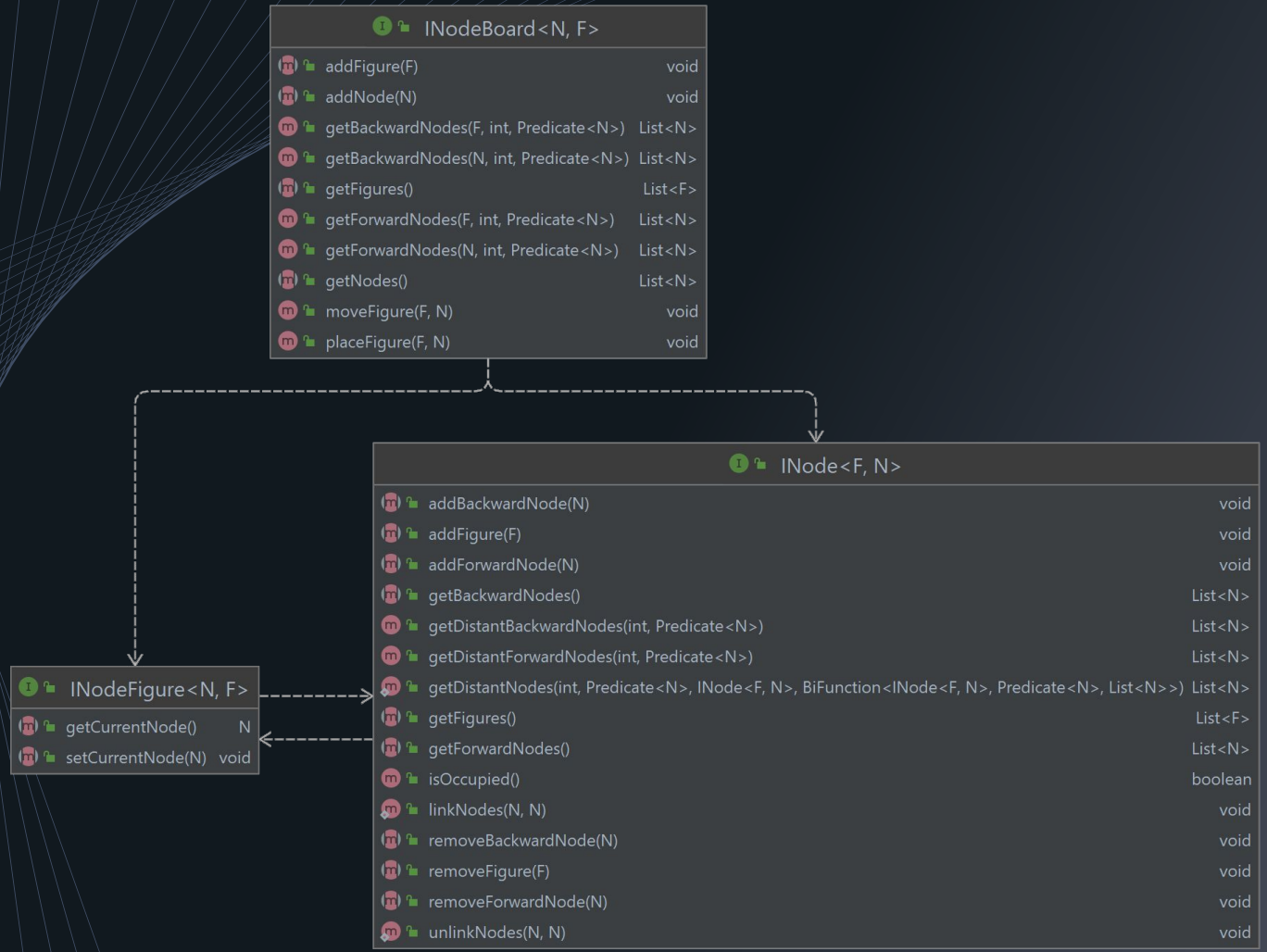
Nodes

- Board as graph like structure
- Nodes linked to neighbours

Alternative: Array or matrix

- Fields structure as array
- For games like chess




INodeBoard<N, F>
addFigure(F) void
addNode(N) void
getBackwardNodes(F, int, Predicate<N>) List<N>
getBackwardNodes(N, int, Predicate<N>) List<N>
getFigures() List<F>
getForwardNodes(F, int, Predicate<N>) List<N>
getForwardNodes(N, int, Predicate<N>) List<N>
getNodes() List<N>
moveFigure(F, N) void
placeFigure(F, N) void
INode<F, N>
addBackwardNode(N) void
addFigure(F) void
addForwardNode(N) void
getBackwardNodes() List<N>
getDistantBackwardNodes(int, Predicate<N>) List<N>
getDistantForwardNodes(int, Predicate<N>) List<N>
getDistantNodes(int, Predicate<N>, INode<F, N>, BiFunction<INode<F, N>, Predicate<N>, List<N>>) List<N>
getFigures() List<F>
getForwardNodes() List<N>
isOccupied() boolean
linkNodes(N, N) void
removeBackwardNode(N) void
removeFigure(F) void
removeForwardNode(N) void
unlinkNodes(N, N) void
INodeFigure<N, F>
getCurrentNode() N
setCurrentNode(N) void

Further Important Classes

Dice

- Generic dice implementation
- Custom faces and face number

TeamColor

- TeamNode
- TeamNodeFigure
- Nodes and figures with team and color

General Game Implementation

- Board
  - Stores current game state
- Screen
  - Displays board and subscreens
- GameLogic
  - Controls the gameflow
- MVC Pattern

Internationalization

- Multi-Language Support
  - English
  - German

- Language set in Json-file
  - Loaded via Gson
  - Input key to get localized output
    (eg. “menu.main.connect” -> “Spiel beitreten”)

Networking FW

About Networking

- Java Sockets (TCP)
- Packet/Message based
- Host-Client network
  - The host is also a client!
- No dedicated server
- Internet + LAN
- Very easy usage!

Host vs. Client

Host

- Integrated server
- No message delay
- Messages to integrated server are run immediately (also for server → host-client)

Client

- Connected to server (host)
- Message delay (ping)
- Messages to server are encoded, decoded, then run (also for server → client)

→ Host and client are only different in what interface they used to send messages to the server; otherwise their client-code is the same!

Simple vs. Advanced Networking

Simple Networking Protocol

- Connect to or host a server
- Send messages
- Receive and run messages
- Close the connection
- Know when a client has connected

Advanced Networking Protocol

- Simple networking included
- Authentication + Player names
- Know when the other side closed their connection
- Know when the connection has been lost (heartbeating)
- Know when you got kicked
- ...

Example Usages

```
hostManager = NetworkHelper.advancedHost(protocol, hostEventsListener,
                LudoClient::new, playerName, port);
hostManager.sendMessageToAllClients(new ChatMessage(client.getName(), text));
hostManager.kickClient(client, "No cheating!");
```

```
clientManager = NetworkHelper.advancedConnect(protocol, clientEventsListener,
                  playerName, ip, port);
clientManager.sendMessageToServer(new ChatMessage(inputWidget.getText()));
```

Open Source!

https://github.com/CAS-ual-TY/plus-board-game-framework