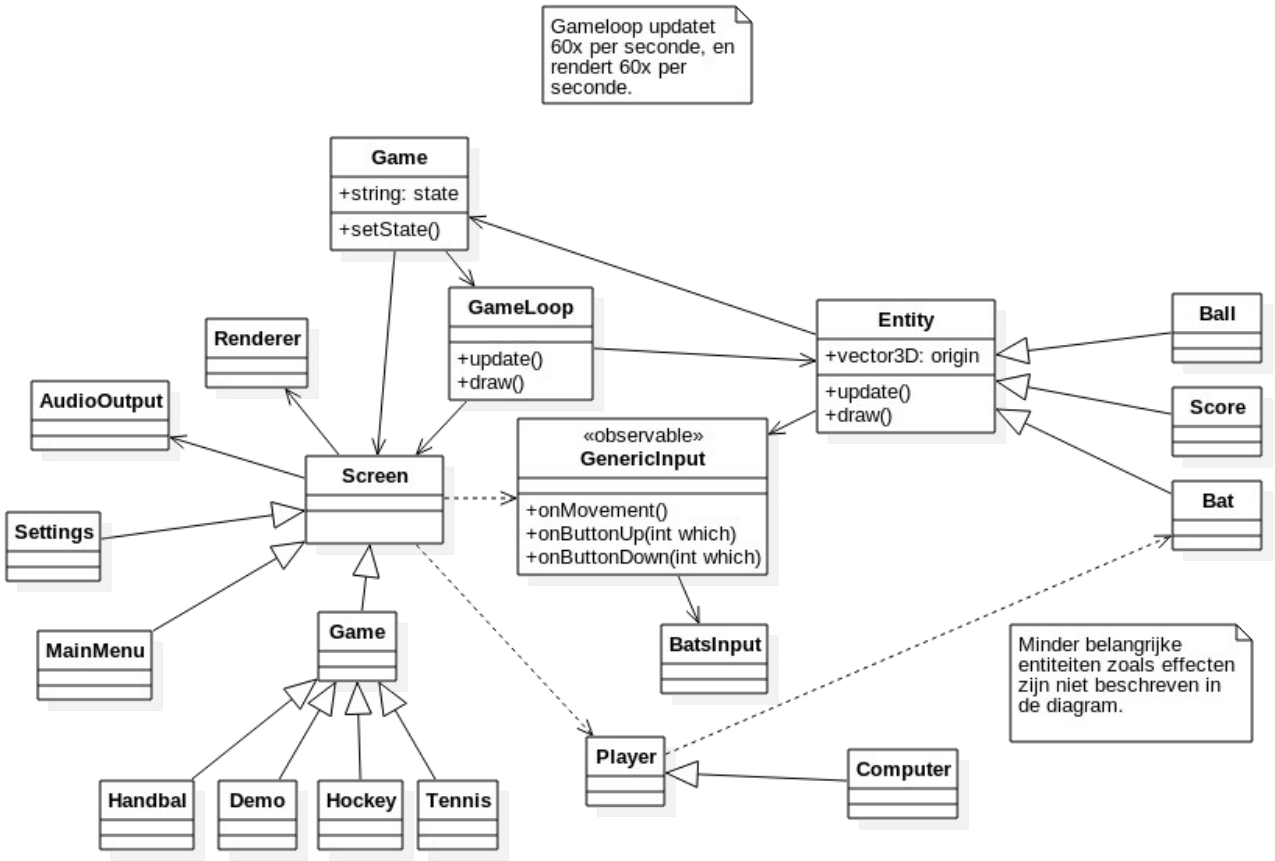
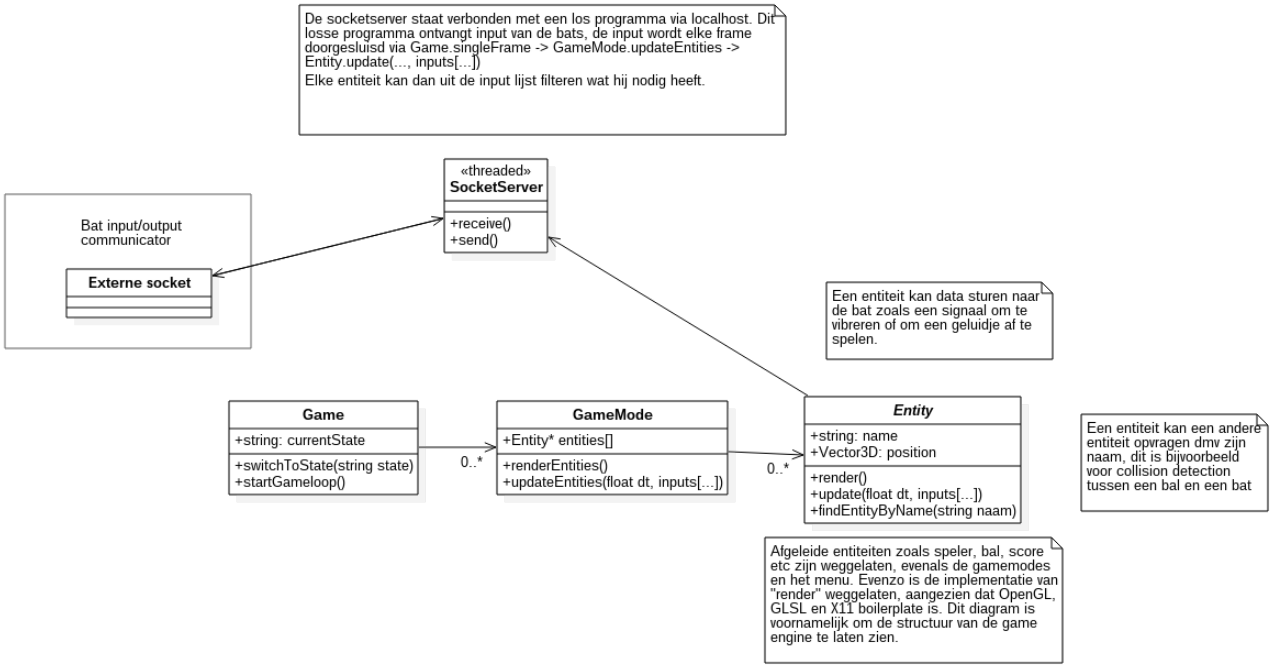
OOAD3 – Ontwerp Game Engine project 3D pong
Martijn Brekelmans, 2072491



Origineel ontwerp game-engine (inclusief game implementatie)



Nieuwst ontwerp game-engine, op basis van het uiteindelijke product (exclusief game implementatie)

OOAD3 – Ontwerp Game Engine project 3D pong
Martijn Brekelmans, 2072491

Uitleg

De architectuur is ontworpen om het zo makkelijk mogelijk te maken om nieuwe gamemodes toe te voegen. De filosofie is om achter de schermen zo veel mogelijk werk te doen, zodat het later makkelijk is om nieuwe entiteiten en gamemodes te ontwerpen en toe te voegen.

Het ontwerp is los gebaseerd op een Entity-based game-engine. Ik zeg los, omdat ik me niet veel heb verdiept in dat type game engines.

Het nadeel van dit ontwerp is dat het grootste en moeilijkste deel van de code “achter de schermen” is. Dit zijn grote stukken boilerplate code om GLSL shaders in te laden, geometrische OpenGL vormen te definieren die gesubclassed kunnen worden door entiteiten, sockets om te communiceren met het tweede scherm en de bats, en als laatste code om een X11 scherm aan te maken. Al deze code vormt het grootste deel van de source code, maar maakt voor het ontwerp weinig uit. Het is code die een keer geschreven hoeft te worden, waarna er eigenlijk nooit meer naar gekeken hoeft te worden. Onder de grond zijn deze stukken code erg complex, terwijl wat er blootgesteld wordt als API iets is als `createWindow()` of `createContext()`. Deze code wordt allemaal een keer geincludeerd, hun functie wordt aangeroepen, en er wordt daarna niet meer naar omgekeken.

Nadelen en verbeteringen:

Er is veel coupling tussen verschillende klasses, bijvoorbeeld bij de genericInput en GameLoop klasses. Dit is nadelig voor eventuele latere uitbreiding en voor het onderhoud van de structuur omdat het refactoring moeilijker maakt. Ook is er een directe coupling tussen alle entiteiten en de output die afgespeeld wordt. Zo staat er op dit moment in Ball.cpp een stukje logica om de juiste code door te sturen naar de fysieke bat om een geluidje af te spelen wanneer hij een ingame bat of muur raakt. Wat een mooiere architectuur zou zijn is een event-based messaging systeem, waarbij een entiteit een signaalt kan sturen, waarop andere objecten dan kunnen reageren.

Bijvoorbeeld:

Ball.cpp

```
…
if (hitWall(this)) {
    messages.emit(“hitWall”);
}
…
```

Sounds.cpp

```
…
void hitWallSound() {
    bat.send(CODE_PLAY_SOUND_SOFT);
    bat.send(CODE_VIBRATE_BAT_SOFT);
}

void soundInit() {
    messages.subscribe(“hitWall”, hitWallSound);
}
…
```

OOAD3 – Ontwerp Game Engine project 3D pong
Martijn Brekelmans, 2072491

Het grote voordeel van een event-based messaging systeem als dit is dat logica die niet thuishoort in een entiteit losgekoppeld kan worden van die entiteit. Ook is het makkelijk uitbreidbaar omdat alles wat je hoeft te doen om nieuw gedrag toe te voegen is een messages.subscribe call, en er niet veel geknoeid hoeft te worden met header files. Het maakt de message klasse ook niet uit vanuit waar hij wordt aangeroepen, of het nou vanuit een entiteit is of vanuit een los stuk code als sound.cpp

Wel heeft het een nadeel voor de leesbaarheid van de code, aangezien de programmeur een web van emits en subscribes moet bijhouden in zijn hoofd. Dit maakt alleen niet veel uit bij een kleinschalig project als dit.