Git - skriftlig redovisning

✨ Fabulous Stars ✨

Ben Bright <yhmu22.brigbe@folkuniversitetet.nu>

2022-12-22

Github: https://github.com/alkafri/fabulous-stars-hangman/tree/release_fixes2

Innehåll

Inledning

Vi var en grupp av fyra personer som med hjälp av kundens vision skulle skapa ett hänga-gubbe-spel åt kunden. För att åstadkomma detta skulle vi jobba Agilt, använda Github och Trello som verktyg och skriva spelet i Java/JavaFX.

- Github: https://github.com/alkafri/fabulous-stars-hangman
- Trello: https://trello.com/b/5cNmuy2D/fabulous-stars-hangman

Vision

Våran tolkning av kundens vision kan sammanfattas med följande: Kunden ville ha ett spel där minst två spelar kan samtidigt spela mot varandra med var sin gubbe utan att behöva vänta på varandra när man chansar på ord.

Första kundmötet

Vid första mötet försökte vi hitta ramarna och kraven för spelet. Vi kom fram till att vi skulle utveckla ett nätverksbaserat spel – en praktisk lösning då man spelar flera och mot varandra.

Vi pratade också om temabaserat där man kanske istället för att hänga gubbe, kunde t ex sänka gummibåt eller krascha raket, men detta hade lägre prioritet inför nästa möte där vi skulle visa upp en prototyp för kunden.

Andra kundmötet

Gruppen kunde dessvärre inte visa upp en prototyp vid detta tillfälle, dels på grund av olyckliga omständigheter, men också på grund av att ”spelmotorn” blev komplex och krävde mer tid för att färdigställas. Lyckligtvis var kunden förlåtande denna gång, men med ett definitivt krav på en fungerande prototyp till nästa möte.

Tredje kundmötet

Vi lyckades ta fram en spelbar prototyp där spelare spelar mot varandra över nätverk som vi visade upp för kunden. Gruppmedlemmarna och kunden spelade några omgångar av spelet. Gruppen upplevde att kunden totalt sett var nöjd med produktprototypen.

Reflektion / User Stories

”Create interface mockup”

Deltagare

Ahmad, Ben, Julius, Manjurekha

Formulering

Formuleringen är kort och kanske inte helt självklar. Gränssnitt kan vara interfaces i kod eller grafik. I detta fall handlade det dock om spelets UI.

Tillvägagående

Vi bestämde oss för att alla i gruppen skulle vara delaktiga i att skissa på ett UI. Sedan skulle vi jämföra och anamma det som var bra. Själv använde jag SceneBuilder för att skissa lite på en JavaFX UI. Manjurekha gjorde någon user input skiss i kod. Julius och Ahmad jobbade på kanvas innehållet med hänga-gubbe grafik och layout.

Det slutade med att vi fick till en UI grund att bygga vidare på.

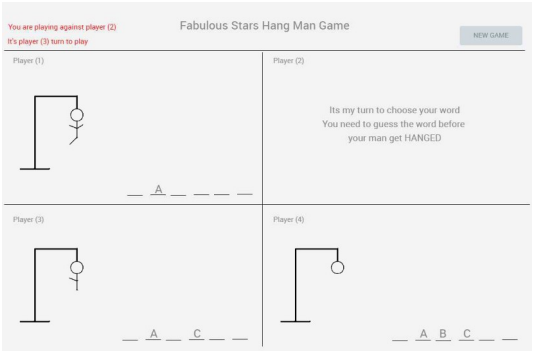
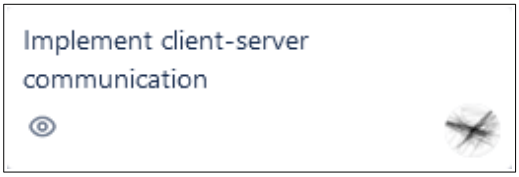


Figure 1: Canvas skisser

”Implement client server communication”



Deltagare

Ben

Formulering

Uppgiften var en ganska stor del av projektet som gjorde att formuleringen blev något luddig. Man borde ha delat upp det i delar då klienten har en GameManager som pratar med GameServlet i servern.

Tillvägagående

Jag ville skapa en kommunikationskomponent i spel-klienten − IGameManager implementationen − som hanterade synkroniseringen av spelet med backend sidan samt server delen. Det fungerar så att spelklienten sätter en eventhandler i GameManager som sedan får alla spel händelser från denna. GameManager pollar servern och får nya händelser som GameServlet upptäcker i händelse-kön, i databasen. Händelserna skickad som serialiserade GameEvent objekt över HTTP i GET kroppen. GameManager genereras sedan händelse objekt som skickas till den registrerade handlern.

```
protected void poll(RequestContext ctx) throws IOException {
    // set body type
    ctx.resp().setContentType("application/octet-stream");
    // get reader
    var output : ServletOutputStream = ctx.resp().getOutputStream();
    // query oldest first
    var entityIter : Iterator<Entity> = datastore.prepare(
            new Query(EVENT_TYPE)
                    .setFilter(new Query.FilterPredicate(
                            propertyName: "target", Query.FilterOperator.EQUAL, ctx.session())
                    )
                    .addSort( propertyName: "expires", Query.SortDirection.ASCENDING)
    ).asIterator();
    if (entityIter.hasNext()) { // get one
        var entity : Entity = entityIter.next();
        var bytes : byte[] = EntityUtils.getBlobBytes(entity);
        datastore.delete(entity.getKey());
        output.write(bytes);
    } else { // no events. send idle
        // empty
        var idle : byte[] = ObjectHelper.toBytes(new GameEvent(GameEventType.Idle));
        output.write(idle);
    }
```

Figure 2: BaseServlet.poll()

För att få detta att fungera fick jag klura ut alla olika händelser som behövdes skickas till klienten (GameEventType). Eftersom GameEvent är en serialiserad objekt så lade jag till en payload instansvariabel i klassen. Här kunde jag till exempel lägga till en lista av PlayState objekt (state för en spelare) för en GameEventType.Play_State event.

```
private boolean threadPoll() {
    try {
        var req : HttpRequest = HttpRequest.newBuilder(new URI( str: backendUrl + "/api/poll"))
                .GET().build();
        var resp : HttpResponse<byte[]> = http.send(req, HttpResponse.BodyHandlers.ofByteArray());
        if (resp.statusCode() >= 400) {
            sendEvent(new GameEvent(GameEventType.Reset));
            return true;
        }
        var event : Object = ObjectHelper.fromBytes(resp.body());
        if (event != null) {
            if (!((GameEvent) event).getType().equals(GameEventType.Idle)) {
                sendEvent((GameEvent) event);
                return false;
            }
        }
    } catch (ConnectException ex) {
    } catch (Exception e) {
        e.printStackTrace();
    }
    return true;
}
```

Figure 3: GameManager.threadPoll()

”Create player canvas component”

Deltagare

Julius

Formulering

Julius skulle skapa en kanvas komponent som ritar upp kanvas innehållet baserat på spelarens tillstånd.

Tillvägagående

Julius skapade arrays correctGuesses och wrongGuesses. Innehållet i correctGuesses sattes till ’*’ för alla bokstäver och wrongGuesses ökades på ned fel gissningar. Dessa ritades sedan upp på en JavaFX Canvas med hjälp av dess GraphicsContext.

Hänga-gubben grafiken ritades upp som färdiga bilder men drawImage() medan text ritades med dtawText().

Under utvecklingsstadiet hade vi spellogik i dessa metoder för att Julius skulle kunna testköra, men när det blev färdigt så flyttades logiken till servern. Kanvas metoderna fick nu all färdig data den behövde för att rita upp kanvasen.

Avslutning

Gruppmöten bestod av att ge uppdateringar om våra uppgifter, eventuella förändringar som behövde göras, samt fördelning av nya uppgifter. Vi hade inga specifika roller tilldelade i gruppen, men jag tyckte att Ahmad rörde sig mot produktägarrollen. Resten av oss var de ”vanliga” utvecklarna.

Av någon anledning hade våran grupp svårt för att använda Trello/Kanban. Det skrevs väldigt lite där. Själv använde jag mest issues systemet på Github för att deklarera mina sprints, men skrev vissa lite luddiga kort i Trello. Här behövs helt klart bättring. Jag tror att hade vi haft någon som tog an rollen som produktägare och verkligen kollade backlog så skulle vi haft en mycket bättre projekt-översikt och prioriteter.

Våra kodgranskningar var inte så avancerade. Resten av gruppen kunde inte granska min kod eftersom jag ligger lite före i Java. Jag gav någon kommentar när det gällde hur man kommenterar kod, men inte mer än så. Jag upplevde inga problem i gruppdynamik från detta.

Det mest svårhanterliga problemet tyckte jag var att felsöka klient-server kommunikationen. Detta eftersom IntelliJ av någon anledning inte fångade breakpoints. Jag kunde således inte steppa igenom kod som skulle ha hjälpt oerhört mycket.

Et annat problem var att, för min del, hantera tiden. Jag tog på mig ganska mycket jobb som krävde tid.

Suck