Project GSO: Walkthrough

Om jullie wat op weg te helpen, geven we in dit document aan hoe de opgave stap voor stap kan uitgewerkt worden. Wat volgt is bedoeld als een hulpmiddel en hoeft zeker niet strikt gevolgd te worden.

1 Opgave doornemen

1. Bekijk de slides voor een initieel overzicht
2. Probeer de concepten zoals het Entity System Framework en Inversion of Control (IoC) goed te begrijpen.
3. Lees de uitgeschreven opgave voor verdere details
4. Bekijk de extra bronnen:
    a. Java Streams: https://www.youtube.com/watch?v=rVfRDLIw_Zw
    b. Entity System Framework: http://www.richardlord.net/blog/what-is-an-entity-framework

2 Architectuur

Vooraleer je start met de implementatie is het interessant om eerst een conceptueel beeld te vormen van hoe je denkt dit spel te kunnen uitwerken. Zijn er design patterns waar je onmiddellijk aan denkt? Wat stel je jezelf voor bij een Entity System Framework, hoe denk je op modulaire wijze de game concepten te kunnen opsplitsen?

In essentie zijn we in deze fase dus de architectuur voor dit project aan het ontwerpen. In principe hoort hier dus een volledige uitwerking van de verschillende softwarecomponenten bij, met de nodige UML-diagrammen, etc… Het is echter niet evident om uit het niets een mooie, consistente architectuur te bedenken. Daarom stellen we jullie voor een meer iteratieve benadering toe te passen:

- Denk na over hoe je een bepaald probleem zou kunnen benaderen. Iets grafisch voorstellen kan helpen (al dan niet UML) om tot nieuwe inzichten te komen.
- Toets je oplossing aan de werkelijkheid door een eerste implementatie te maken. Deze mag nog wat slordigheden bevatten en eventueel bepaalde zaken suboptimaal oplossen.
- Probeer te leren uit deze eerste implementatie. Wat is er goed aan? Wat werkt er niet zoals je gedacht had? Dit is nuttige informatie om je theoretische oplossing (je architectuur) mee bij te sturen.
- Soms kan het zijn dat je moet zoeken naar een compleet andere benadering (bv. een hypothetisch algoritme voor pathfinding heeft een rekentijd van minstens 200 ms, waardoor deze onbruikbaar wordt in een game loop). Als de initiële implementatie wel voldoet, kan je de architectuur verder uitwerken en daarna aan de slag gaan met een tweede versie van de implementatie.
- Bovenstaande stappen kunnen herhaald worden tot de oplossing volledig functioneel is en ook bijkomende niet-functionele vereisten voldaan zijn (zoals een voldoende snelle reactie-tijd).

In het verslag bij het project zal je uitleg moeten geven bij de belangrijkste ontwerpsbeslissingen en het gebruik van design patterns, dus het werk dat in deze stappen gebeurt, naast het effectief programmeren, is sowieso niet verloren.

Als referentiekader geven we mee dat onze voorbeeldoplossing gebruik maakt van de volgende design patterns:

- Singleton
- Flyweight
- Strategy
- Observer
- Decorator
- Factory
- Template method

Dit betekent niet dat ook de andere geziene patronen niet nuttig kunnen zijn of dat jullie verplicht zijn elk van deze patronen ook te gebruiken in jullie eigen implementatie.

3 Implementatie

3.1 Eerste Stappen

3.1.1 Opbouw Game structuur

De opgave geeft een introductie tot game development in Java FX 8, maar hoe zal je dit integreren met het Entity System Framework waarop het spel gebouwd zal worden?

Probeer ook al eens te experimenten met een Canvas te laten zien op het scherm en kijk of je er iets op kan tekenen!

3.1.2 Eerste systemen

Nu alles op z'n plaats staat, kan je eindelijk aan de slag gaan met het Entity System Framework. Begin met bijvoorbeeld een player entiteit aan te maken en registreer twee system die erop inwerken:

- **MovementSystem**: past elke iteratie de TransformComponent aan, zodat de speler over het scherm beweegt.
- **RenderSystem**: tekent de speler naar het scherm op een vaste achtergrond.

3.1.3 Player Movement

De volgende stap is zelf de controle over de speler overnemen aan de hand van de input controls. Je dient dus een manier te vinden om te luisteren naar keyboard en mouse input (via Canvas) en dit om te zetten naar commando's voor het Entity System, bv. de WASD-toetsen zullen leiden tot een specifieke bewegingsvector voor de speler, terwijl de muis de kijk- en schietrichting van de speler zal bepalen.

Vergeet niet dat je hiervoor extra Componenten en Systemen zal dienen te introduceren! Probeer hierbij een losse koppeling in te bouwen tussen enerzijds het afhandelen van de keyboard en mouse input events en anderzijds het genereren van de movement vector en andere speler acties. Bijvoorbeeld:

1. Vanuit Game reageer je op de JavaFX handlers voor muis en keyboard input, bv. door een Entity te maken die de toestand van de input combineert en deze toe te voegen aan de Engine.
2. Je kan een system klasse bouwen die zich specifiek de componenten van de input Entity analyseert en op basis hiervan bv. een TransformComponent toevoegt aan de speler entity.

3. Je MovementSystem kan je op deze manier generiek houden doordat het gewoon TransformComponents toepast op enemy of speler entiteiten los van of deze gegenereerd zijn door een algoritme of door input.

3.1.4 De camera

In de demo is te zien dat de wereld eigenlijk groter kan zijn dan wat er op het scherm te zien is. We lossen dit op door een "camera" te introduceren die zich focust op de speler. De speler zal dus altijd centraal staan, tenzij hij zich naar de randen van de spelwereld begeeft.

Denk eens na over hoe je dit zou kunnen implementeren!

3.2 Het eigenlijke spel

Als alles correct verlopen is, hebben we nu een canvas waarbinnen we met een speler kunnen rondlopen en kijken. Om er nu een echte survival shooter van te maken, hebben we nog drie zaken nodig: een wapen, vijanden en in mindere mate een animatie-systeem.

3.2.1 Eerste wapen & projectielen

Het eenvoudigste is om te beginnen met een gewoon pistool (de "handgun") te implementeren. Deze schiet één projectiel per shot af, dat in een rechte lijn in de kijkrichting van de speler vliegt. Na een bepaalde afstand laat je het projectiel uiteindelijk verdwijnen, dit is het bereik van het wapen.

Denk na over de typische karakteristieken van een wapen en hoe je dit kan modeleren, ook met het oog op de latere uitbreidingen, zoals bijvoorbeeld het toepassen van bonussen.

3.2.2 Eerste vijand

Neem niet teveel hooi op jullie vork, ga van start met één enkele vijand in de wereld te plaatsen op een vaste locatie in de buurt van de speler. Probeer deze te laten bewegen in een bepaalde richting. In een volgende stap kan je uitzoeken hoe je een vijand een bepaald aantal levens kan geven en deze dan ook kan verliezen wanneer het geraakt wordt door een projectiel van de speler.

Voor dit laatste zal je een manier nodig hebben om aan collision detection te doen. Collision detection is het zoeken naar entiteiten die overlappen en dus in een botsing verwikkeld zijn. De eenvoudigste oplossing is om een vierkant van een bepaalde grootte (voorgesteld als een Rectangle2D) aan elk van de entiteiten waarvoor dit relevant is te hangen. Op deze manier kan je elke iteratie van de game loop nagaan of er doorsnedes zijn voor bepaalde van de vierkanten. Typisch maak je deze zogenaamde "hitbox" kleiner dan de effectieve sprite waarmee je de entiteit een visuele weergave geeft, dit om wat speling te hebben op wanneer de entiteit iets raakt.

Eens de vijand door genoeg projectielen geraakt is en dus geen levens meer heeft, dien je deze in een gedode toestand te zetten. Je kan dit op verschillende manieren aanpakken, maar het resultaat is steeds dat de vijand van het scherm verdwijnt en vervangen wordt door een plas bloed.

Nu de vijand kan rondlopen en gedood worden, is het moment aangebroken voor iets van dreiging te zorgen voor de speler. Implementeer hiervoor een mechanisme dat de vijand steeds in de richting van de speler laat lopen. Opnieuw kan je collision detection gebruiken om te zien of de vijand de speler uiteindelijk raakt. In dit laatste geval dien je schade toe te brengen aan de speler.

3.2.3 Animaties

We hebben nu reeds een werkend spel, maar wel ééntje die er wat saai uitziet. Animaties kunnen hierbij helpen. Een animatie in een sprite-based 2D spel is eigenlijk een collectie van meerdere sprites die in een bepaalde volgorde gealterneerd worden, zodat je de indruk krijgt dat je naar een object in beweging aan het kijken bent (cfr. GIF-figuren).

Denk eens na over hoe een animatie-systeem zou kunnen werken. Begin daarna met de animaties uit te werken voor de vijand (het type soldier beschikt slechts over vier frames). Als dit vlot gaat, zou je ook al kunnen de speler animeren.

3.3 Het spel uitbreiden

In een volgende stage kunnen we de basis van het spel uitbreiden met de overige spelelementen die de gameplay interessant zal maken.

3.3.1 Overige wapens

Implementeer de shotgun en rifle varianten en test deze uit door ze tijdelijk als standaard wapen in te stellen voor de speler.

- **Shotgun**: vuurt geen kogel af, maar een reeks hagel die zich op willekeurige wijze verspreid over een korte afstand.
- **Rifle**: vuurt meerdere kogels per shot af, met elk een kleine willekeurige afwijking.

Implementeer als laatste de grenade launcher variant, deze is iets complexer aangezien het hier niet gaat om een projectiel dat rechtstreeks schade aanricht, maar om een granaat die bij impact een explosie zal veroorzaken. Denk na over hoe je dit zou kunnen doen.

Tip: een bepaald gedrag voorgesteld in code, kan ook als data aan een component gehangen worden. Bestudeer hiervoor de standaard Java functionele interfaces Consumer en BiConsumer die ook een extra attribuut voor je component definities kunnen zijn.

3.3.2 Overige vijanden

Tot nu toe biedt het spel niet echt een uitdaging daar er maar één enkele vijand aangemaakt wordt. Er is nood aan een mechanisme dat over de nodige logica beschikt om te beslissen wanneer er nieuwe vijanden worden aangemaakt, hoeveel er dit moeten zijn en van welk type. Dit mechanisme zal er ook voor zorgen dat de vijanden op willekeurige plaatsen binnen de spelwereld worden aangemaakt.

Tip: logaritmische functies zijn goed geschikt om iets geleidelijk aan moeilijker te maken.

3.3.3 Bonussen

Nu er horden aan vijanden in de wereld geplaatst worden, zullen we de speler wat moeten helpen bij het proberen te overleven. Dit doen we door bonussen te introduceren, die met een bepaalde kans tevoorschijn komen wanneer de speler een vijand kan doden. Denk na over hoe je dit kan implementeren.

Tip: misschien kan de manier waarop je de explosie voor de grenade launcher geïmplementeerd hebt, ook hier van toepassing zijn.

Nieuwe wapens die door de speler kunnen opgeraapt worden, kunnen ook als bonus item beschouwd worden.

3.3.4 Level-up system

Om een level-up systeem te implementeren, zal enerzijds XP moeten toegekend worden aan de speler en is er een manier nodig om te bepalen wanneer de speler precies in level stijgt. Denk ook hier opnieuw aan een logaritmische functie.

Daarnaast moeten er ook upgrades kunnen toegekend worden bij een level-up. Misschien kunnen deze ook voorgesteld worden als bonussen?

3.4 Afwerking

Na sectie 3.3 hebben we al een vrij functioneel spel dat al wat mogelijkheden biedt. Nu resteert er ons eigenlijk alleen nog de afwerking.

3.4.1 UI Elementen

Het is handig om de speler ietwat informatie te geven over de verschillende spelaspecten, zoals bijvoorbeeld:

- Weergeven van het huidige wapen van de speler en hoeveel kogels er nog resteren in het magazijn.
- Weergeven van de hoeveelheid XP dat nog nodig is om een volgend level te bereiken.
- Weergeven van over hoeveel levens de speler nog beschikt en wanneer een schild van toepassing is.
- Weergeven van hoeveel vijanden de speler tot nu toe gedood heeft
- Weergeven welke bonussen er momenteel actief zijn
- Weergeven over hoeveel van elke level-upgrade een speler al beschikt
- …

Deze zaken zullen als laatste op het scherm getekend worden en verschillen van de overige game entities doordat hun positie relatief is t.o.v. het scherm en niet t.o.v. de achterliggende wereld. In deze zin kan de crosshair van de speler ook als een UI element beschouwd worden.

3.4.2 Geluid

De opgave bevat naast de sprites ook een aantal geluidsbestanden die je zou kunnen afspelen op de geschikte momenten om het spel nog iets meer leven in te blazen. Implementeer dit aan de hand van een nieuwe Component en System klasse.

3.4.3 Gameplay tweaken

Dit houdt in dat je met bepaalde parameters zal spelen en nagaat welk effect deze hebben op de gameplay. Het doel hiervan is om je spel uitdagend genoeg te maken, zonder dat het onmogelijk wordt. Eventueel kan je verschillende moeilijkheidsgraden introduceren die kunnen ingesteld worden via de config.