Technisch Verslag Thema opdracht Game

Net Attack

Stefan de Beer // 1684137
Paul Ettema // 1677247
Jeroen Kok // 1666479
Brian Duncker // 1677303
Bart van der Kolk // 1663533
Amrit Malhi // 1691806

1/2/17 // V0.1

Index

Inleiding

Dit spel is opgesteld als thema opdracht gedurende het tweede jaar van de studie Technische Informatica. De opdracht was om in teamverband een spel te creëren volgens de Scrum methode. Het project maakt gebruik van CMake en is geschreven in C++.

Het doel van het verslag is om de developers die gebruik willen maken van dit project, als basis voor hun eigen project, een leidraad te geven in hoe het project is opgebouwd. Met deze informatie moet het mogelijk zijn om verder te werken aan dit project.

Functioneel ontwerp

Features

- Vier speelbare karakters.
- Kan singleplayer en coop worden gespeeld.
- Een wereld waarin alle nodige karakters tegelijk in spelen.
- 2D Side Scrolling Platformer.
- Een aparte story voor elk karakter, die langzaam samenkomen tot één groot geheel.
- Puzzels oplossen om de levels te beïnvloeden om het level te behalen.

Speler Motivatie

Je bereikt het einde van het verhaal door verschillende uitdagende puzzels op te lossen.

Genre

De genre is een 2D platformer en puzzel game.

Mechanics

In het spel kun je soms pas ergens voorbij komen als dat deel veranderd is door de acties van een ander karakter met een speciale ability. Deuren en platforms kunnen worden veranderd door logische poorten en schakelingen op te lossen.

Unique Selling Points

- Wat een karakter verandert in de wereld, verandert ook voor de andere karakters.
- Multiplayer.
- Uitdagende minigames.
- Multiple rooms.
- 2D side on.
- 4 player co-op.
- Meerdere unieke karakters met speciale vaardigheden.
- Pixel graphics.

Doel Platform

Platform: PC met ondersteuning voor Windows en Linux.

Ontwerpdoelen

- De puzzels moeten uitdagend zijn.
- De veranderingen aan de wereld door de andere karakters moeten duidelijk merkbaar zijn.
- Een simpele physics engine voor de karakters.
- Meerdere werelden, die levels hebben, die op hun beurt rooms hebben die met elkaar verbonden zijn.

Technisch Ontwerp

De architectuur van het project berust op het idee dat alles in het spel een object is. Een object kan moveable, collisionable, controllable of drawable zijn. Het object kan ook meerdere van de opgesomde zijn. Om een voorbeeld te noemen; een character is moveable, collisionable en controllable. Dit betekent dat een character kan bewegen, kan botsen met andere objecten en bestuurd kan worden door de speler. Hierdoor is het gemakkelijk om een veelzijdig element toe te voegen.

Een level is opgebouwd uit een set van bepaalde soorten objecten die uitgelezen worden uit een tekstbestand(.txt). In dit tekstbestand definieer je het soort object dat je wilt hebben gevolgd door de: locatie, grootte, maximum horizontale snelheid, maximum verticale snelheid, kleur/opaciteit en textuur van het object. Het soort object bepaalt ook of het object een van de hierboven genoemde eigenschappen bevat.

In de main van het project komt alles bij elkaar. De main zorgt ervoor dat de levels worden uitgelezen uit de tekstbestanden en dan die op het scherm worden geschetst als de speler de keuze maakt in het main menu om het spel te starten. Verder zorgt de main ook voor wat overige dingen zoals het switchen tussen karakters, de achtergrond muziek, het pauze menu voor als je in game bent en de ending credits voor als je klaar bent met het spel.

Realisatie

Om aan de slag te gaan zijn de volgende packages vereist:

- SFML (2.4.1)
- CMake

Verder is het aangeraden om Git te gebruiken voor versiebeheer en een stabiele IDE naar keuze zodat je makkelijker code kan schrijven.

Een ontwerp cyclus gaat als volgt te werk. Na het toevoegen van een bepaalde functie of feature in een .cpp of .hpp bestand in de /src/ map sla je die op. Vervolgens navigeer je naar de build map in het project (als deze niet bestaat kan je een lege aanmaken). Vanuit de terminal run je dan het commando “cmake ..” en daarna “make”. Dit zorgt ervoor dat de makefile wordt aangeroepen en dat de code wordt gecompileerd. Om de code te runnen voer je het commando “./thema-game” uit.

Er is gekozen voor de Personal Computer als platform omdat dit het grootste platform is voor games. Hierbij zullen de bestuur systemen Windows en Linux ondersteund worden.

Elke keer als het level wordt gestart wordt er een zogeheten ‘factory’ aangeroepen. Deze leest een tekstbestand en op basis daarvan maakt hij een level.
Zo kunnen gebruikers ook hun eigen levels maken. Ook is er en levels.txt file waarin alle levels staan. Om een level toe te voegen moet de directory van het level in deze lijst van levels worden geplaatst.

De syntax van een rectangle in een level.txt is:
rectangle positie.x positie.y size.x size.y r g b texture(optioneel).

Van een circle
circle positie.x positie.y size r g b texture(optioneel)

Voor een wall:
wall positie.x positie.y size.x size.y r g b texture(optioneel)

Voor een killbox
killbox positie.x positie.y size.x size.y r g b texture(optioneel)

Van een character*:*
character postitie.x positie.y size.x size.y loopsnelheid spronghoogte r g b texture(optioneel)

Om de bewegingen en tekenen in het spel af te handelen gebruiken wij 2 loops;
De eerste is die alle key-input afhandeld. Er wordt gekeken welke relevante toetsen zijn ingedrukt en daarbij worden functies van character aangeroepen. Bijvoorbeeld als je op de linker-pijl drukt, wordt de snelheid van het character verhoogd.
Daarna wordt er door een lijst met alles wat kan bewegen geïtereerd. Alles wat kan bewegen wordt verplaatst - mocht een object stil staan dan wordt hij niet verplaatst. Hierin

wordt de luchtweerstand, botsingen en zwaartekracht afgehandeld. Deze verplaatsing is nog niet te zien.
De vorige twee loops worden net zolang herhaald totdat een bepaalde tijd is verstreken. Als deze tijd is verstreken zal er worden geïtereerd door een lijst van Drawables. In deze loop wordt alles getekend wat uiteindelijk op het scherm komt te staan.
Door deze verdeling wordt de GPU niet overbelast - enkel 60 keer per seconden. Maar het spel voelt nog steeds responsive, omdat zolang 1/60 van een seconden niet is verstreken, alle input wel wordt afgevangen.

level_button en level_lever werken niet. Om deze werkend te maken moeten ze toegang hebben tot een level object om daar shapes te kunnen veranderen of toevoegen.

Evaluatie

Problemen:

- We hadden van te voren op zijn minst een generiek klassendiagram moeten maken om overzicht te behouden.
- De mogelijkheid was niet opengelaten om buttons en levers te implementeren. Hierdoor is het niet mogelijk geweest om de buttons en levers goed te laten werken. Dit kwam mede door het gebrek aan een klassendiagram en duidelijke documentatie dat van te voren gemaakt had moeten worden om vast te stellen wat precies nodig was.

Oplossingen:

- Het missen van de klassendiagram had opgelost kunnen worden door voor of tijdens de eerste sprint een klassendiagram te creëren en deze actief aan en bij te houden.
- Voor de buttons en levers had er tijdens de tweede sprint reken gehouden moeten worden zodat deze zonder de hele code overhoop te halen te implementeren.

Conclusies en aanbevelingen

Wat zijn de conclusies en aanbevelingen op functioneel vlak?

- Het creëren van een klassendiagram voordat de werkweek begint en met voortschrijdend inzicht brainstormen.
- Betere communicatie over wat sommige functies moeten kunnen leveren om daar andere functies op toe te passen.