Technisch Design

Heimeiden

Thomas van der Berg *Minor Game Design, CMD*

22 Maart 2013 *NHL Hogeschool*

Inhoud

Inleiding

Voor het Fries Museum moet er een Paalwormenspel gemaakt worden. Dit spel zal in een expositie komen te staan over Friesland, in een sectie over de dreiging van water. De speler moet zijn dijk verdedigen tegen een leger van paalwormen die Friesland willen laten overstromen. Het is gebaseerd op Plants versus Zombies. Voor het spel is een functioneel ontwerp en een technisch ontwerp gemaakt. Dit is het technische deel van het ontwerp.

Het doel van dit document is om een overzicht te geven van het spel. Voor specifieke informatie: Kijk in de broncode. Functies die niet overduidelijk zijn in de code hebben een comment erbij die uitleggen wat ze doen.

Korte beschrijvingen

Hardware

Het spel komt op een ingebouwde computer met een 21.5 inch scherm in het Fries Museum. Het komt in landscape opstelling in het museum met 1920×1080 pixels.

Software

De computer waar het spel op komt heeft bevat Windows 7 en Mozilla Firefox. Om het paalwormenspel te produceren wordt gebruik gemaakt van HTML5 technologie met JavaScript. Het wordt gemaakt met een Rendering Engine die eerder al gemaakt is, genaamd HTML5_2DRE. Het spel wordt gespeeld met Firefox in fullscreen.

Rendering Engine

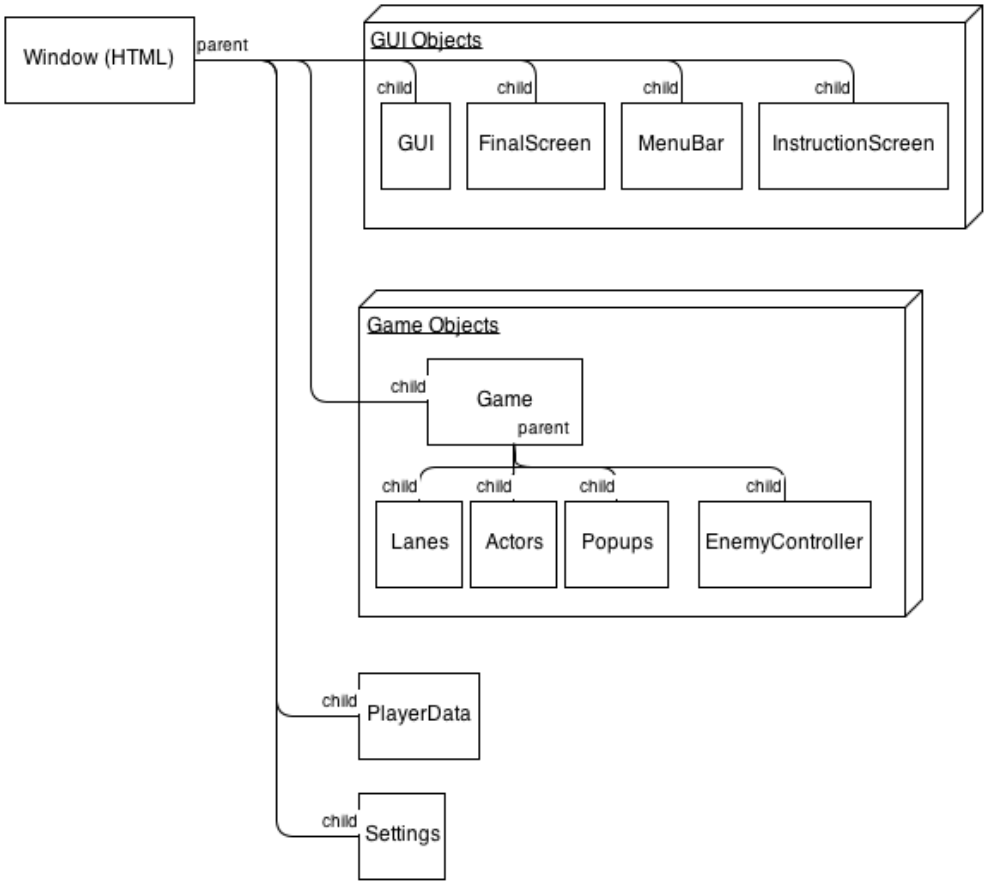
HTML5_2DRE is een rendering engine die voor een vorig HTML5/JS project geproduceerd is om het simpeler te maken om objecten te laten tekenen en game logica te implementeren. De rendering engine regelt voor ons de initialisatie van het renderen, het tekenen van tekst en plaatjes, muis input, en de main update-draw loop. Verder wordt de engine nog uitgebreidt met features die nodig zijn vor het project, zoals animaties en touchscreen input. Touchscreen input wordt geintegreerd met muis input, dat al in de rendering engine zit. De technische documentatie van de rendering engine zou meegeleverd moeten zijn.

Game Engine

De Game Engine wordt intern ontworpen en geproduceerd, en is het hoofdonderwerp van dit document. De game engine bestaat uit een verzameling DrawableObjects(Objecten die inheriten van de Rendering Engines “BaseDrawable” object), losse functies en interacties met JSON-bestanden. Alle game objecten worden beheerd door het object Game, gedefinieerd in heimeiden.js. Alle objecen op het veld inheriten van het object Actor, gedefinieerd in heimeidendrawables.js. Deze objecten hebben dingen als movement, collision detection, health, event functies en interacties met elkaar. Hieruit wordt het spel opgebouwd.

Overzicht

Hierarchische diagram



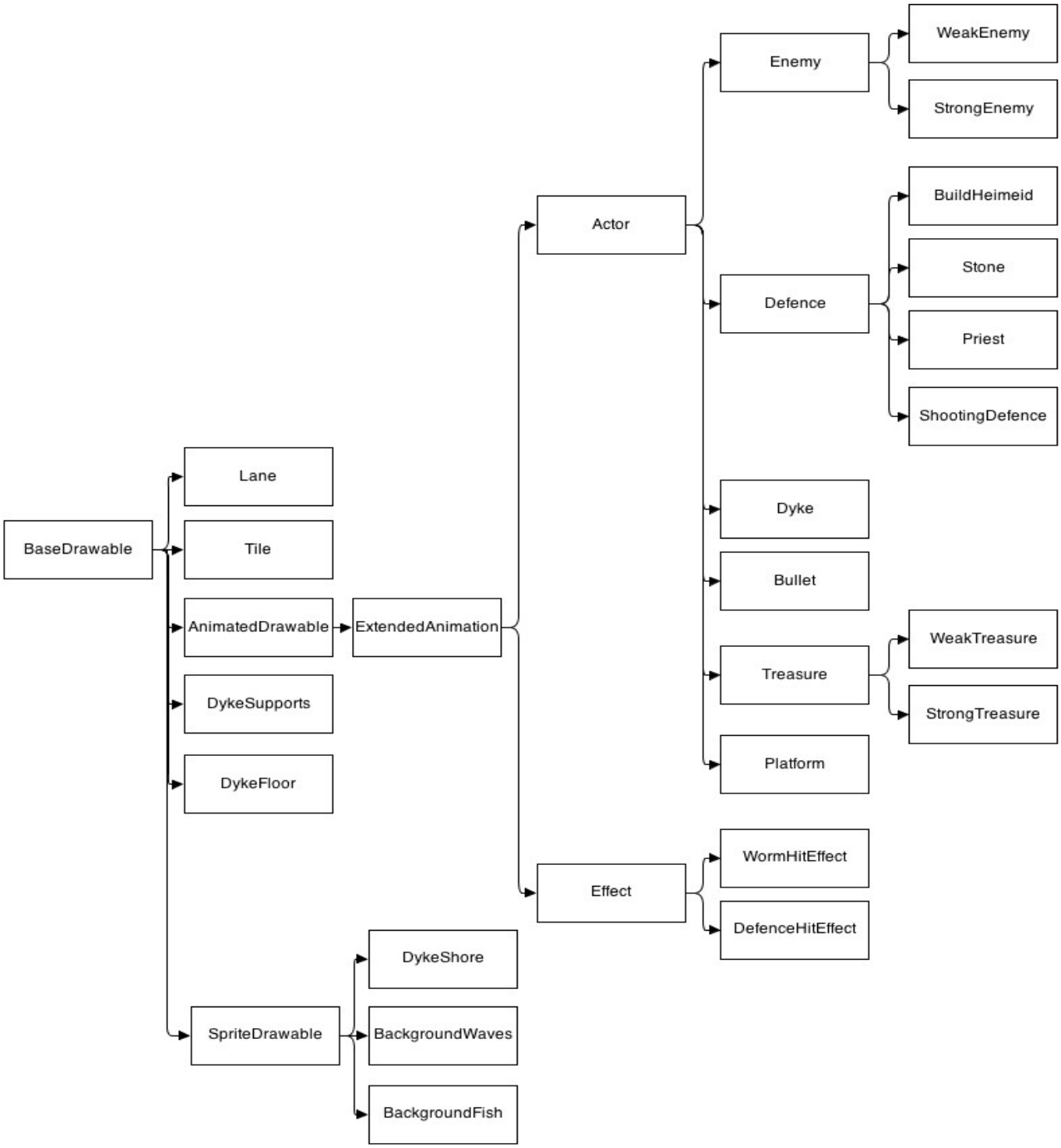
Dit is een overzicht van de hierarchie van statische objecten terwijl het spel actief is.

Lanes is een array van Lane objecten die elk weer hun eigen array van Tile objecten hebben. Zo wordt het veld bijgehouden. Tiles verwerken tevens de input wanner er op het veld geklikt wordt.

Actors is een array waar alle levende actors in staan.

Popups is een array waar alle popup-objecten of effecten op staan. Puur visuele dingen die na een tijdje weggaan.

Inheritance diagram – heimeidendrawables.js



Alles waar een pijl naartoe staat inherit van het object waar de pijl vandaan komt. Bijvoorbeeld: WeakEnemy inherit van Enemy en ExtendedAnimation inherit van AnimatedDrawable.

BaseDrawable is een ingebouwd object van de engine waarvan de update- en draw-functies automatisch worden uitgeroepen als ze toegevoegd zijn aan het spel.

Bestanden

In script/game

audio.js ← Bevat AudioPlayer class

constants.js ← Bevat een aantal constants die in het spel gebruikt worden

enemycontroller.js ← Bevat de enemycontroller class die waves van vijanden beheert via informatie uit waves.json

game.js ← linkt naar de andere bestanden, en bevat een aantal pure functies die gebruikt worden in andere bestanden

getjson.js ← Bevat functies voor het inladen en parsen van lokale JSON-bestanden

gui.js ← Bevat GUI informatie

heimeiden.js ← Bevat het Game-object die de game beheert

heimeidendrawables.js ← Bevat DrawableObject classes: Actors, Lanes, Tiles, etc.

playerdata.js ← Bevat het PlayerData object die speeldata bijhoudt

popup.js ← Bevat functies die popups maakt voor feedback

preload.js ← Bevat functies om afbeeldingen in te laden voor ze gebruikt worden

vector2.js ← Bevat een implementatie van 2D vectors die door het spel gebruikt wordt

In hoofdfolder

settings.json ← grafische en functionele instellingen die gewijzigd kunnen worden, in JSON

waves.json ← informatie over de inhoud van alle waves, in JSON

Andere

script/replaceall.js ← Bevat een externe functie die alle karakters van een bepaald type in een string omzet naar andere karakters

Script/engine/HTML5_2DRE.js ← Bevat de rendering engine

Startprocedure

Het script game.js wordt geinclude in het HTML bestand. game.js zorgt ervoor dat alle andere bestanden in de correcte volgorde worden ingeladen.

Wanneer alles geladen is roept de rendering engine de functie initialize() aan die gedefinieerd wordt in game.js. Deze functie roept de initialize() functie van de class Game aan die in heimeiden.js gedefineerd wordt. Deze initialiseert het spel.

Tijdens initialisatie worden benodigde afbeeldingen ingeladen, de Lanes geïnitialiseerd, de EnemyController geïnitialiseerd, en worden de waarden uit de settings gehaald. Hierna worden de Game en de GUI aan de Model toegevoegd, waardoor de Model dan de update functie zal gaan aanroepen van de Game en GUI. Hierdoor begint de Main Loop.

Main loop

Update functies worden voor elke frame uitgeroepen

Statische update functies:

Game – Voegt credits toe, checkt of de speler gewonnen heeft

EnemyController – Spawnt vijanden en beheert de rondes van vijanden

GUI – Update de teksten

Verder wordt voor veel van de geinstantieerde objecten ook de update-functie uitgeroepen. De belangrijkste hiervan is de update-functie van de Actor class. Deze checkt voor collisions en beweegt de Actors over het veld. Hierdoor worden dan weer andere functies aangeroepen wanneer bijvoorbeeld een paalworm tegen de dijk aanzit, en dit zorgt ervoor dat de Actors bezig gaan en het spel doorgaat.

Input van de speler

De enige input in het spel wordt gegeven via een touchscreen dat een muis emuleert. De muisinput wordt geregeld door de HTML5_2DRE engine. Deze zorgt ervoor dat mouse events(onhover, onclick, onmousedown, etc) bij het correcte object terechtkomen. Zo kan een programmeur voor een object dat van BaseDrawable inherit een onmousedown functie aanmaken die dan uitgeroepen wordt als de speler zijn muisknop indrukt op het object. Verder kan voor elk object de ignoremouse boolean op false gezet worden zodat hij genegeerd wordt bij muis events. Dan gaat wordt de muis event getriggert bij het object dat eronder staat(als die er is).

In het spel wordt deze input gebruikt door knoppen en door de tiles. Alle actors behalve de oppakbare Shell hebben ignoremouse op true zodat een klik doorgaat naar de tiles die het bouwen van verdedigingen regelen.

Eindprocedure

Het spel kan op 3 verschillende manieren eindigen:

- De speler wint
- De speler verliest
- De speler drukt op stop in het zij-menu.

De speler wint wanneer de EnemyController aangeeft dat hij gestopt is, door PlayerData.areWavesFinished op true te zetten, *en* er geen levende vijanden meer zijn. Hierna wordt er een aantal seconden gewacht(aangegeven in settings.timeUntilFreeze), en dan wordt FinalScreen zichtbaar gemaakt met het winscherm, en wordt het wingeluid afgespeelt. De speler kan dan op het winscherm drukken om terug tegaan naar het menu. Het teruggaan naar het menu wordt geregeld door alle data te resetten, alle actors en popups te verwijderen, de ingame GUI elementen onzichtbaar te maken en de splashscreen GUI elementen weer zichtbaar te maken.

De speler wint wanner de dijk is doorgebroken. Net als met het winnen wacht het spel dan voor de tijd aangegeven in settings.timeUntilFreeze, waarna de FinalScreen zichtbaar wordt. Die wordt dan geladen met het verliesscherm en verliesgeluid. Net als in het winscherm kan de speler erop klikken om terug te gaan naar het menu.

Als de speler in het zij-menu op stop drukt, gaat het spel direct terug naar het hoofdmenu. Voor hoe dit wordt gedaan, zie de alinea over wanneer de speler wint.

Aanpassingen maken

Settings of Rondes aanpassen

Het spel bevat twee bestanden waarin het spel aangepast kan worden zonder de broncode in te duiken: settings.json en waves.json.

In settings.json kunnen de waarden aangepast worden van een aantal grafische en functionele onderdelen van het spel. Zo kan het spel gebalancet worden of kunnen dingen aan of uit gezet worden zoals de achtergrondmuziek/geluiden. De meeste van deze waardes spreken voor zich, anders staat er een comment achter die het wat verduidelijkt.

waves.json is iets ingewikkelder. Het bevat alle informatie over de rondes van vijanden die de speler aan komen vallen. Het bestaat uit een array van wave objecten die elk hun eigen informatie bevatten. De wave objecten bevatten een aantal onderdelen:

- 1 of meerdere subwaves met vijanden. Dit is het enige wat verplicht is om in een wave te stoppen. Een subwave is een object met 1 array genaamt “enemies” erin. De eerste waarde van deze array is het aantal normale paalwormen dat in deze wave zit, en de tweede waarde(als die er is) is het aantal sterke paalwormen.
- spawnInterval – Een object dat het minimum en het maximum aangeeft van hoeveel seconden het kan duren tot de volgende vijand gecreeërd wordt. Als deze waarde niet gedefineerd staat, wordt het uit settings.defaultSpawnInterval gehaald.
- waitBeforeWave – Het aantal seconden dat het spel wacht voor deze wave begint. Als deze waarde niet gedefineerd staat, wordt het uit settings.defaultWaitBeforeWave gehaald.
- unlockBuildings – Array van welke verdedigingen er deze ronde geunlockt worden. Als deze waarde niet gedefineerd staat, wordt het uit settings.defaultUnlockBuildings gehaald.
- maxEnemies – Het aantal vijanden dat maximaal in het scherm mogen zijn voor deze ronde. Als deze waarde niet gedefineerd staat, wordt het uit settings.defaultMaxEnemies gehaald.

In dit bestand kun je zo een wave toevoegen, verwijderen of aanpassen. Het spel kan theoretisch oneindig waves aan.