Gate 2158

Team Need-A-Name

Martijn van Dijk (1660713)
Jordan Ramirez (1652760)
Ole Agterberg (1651981)
Christiaan van den Berg (1660475)
Thijs Hendrickx (1660936)

03/11/2015

Versie 0.1

Management Samenvatting

Het doel van de opdracht was om een werkende 2D game te maken, in C++. We hebben daarbij gebruik gemaakt van SFML. De game moest minimaal vijf minuten te speel zijn, er moesten competitive elementen in voorkomen en het moest een uitdaging zijn.

Het doel van dit verslag is om een technisch inzicht te geven van de bouw van de game. Welke keuzes we gemaakt hebben bij het opstellen van het functioneel ontwerp en het technisch ontwerp. Wat onze bevindingen zijn geweest en tenslotte wat onze conclusie en aanbevelingen zijn.

Voor ons functioneel ontwerp hebben we er bijvoorbeeld voor gekozen om de RPG elementen uit de game te laten. En voor ons technisch ontwerp hebben we er voor gekozen om de verantwoordelijkheden van de engine in de klassen zelf te stoppen.

We kwamen er achter dat de keuze om de verantwoordelijkheden van de engine in de klassen zelf te zetten geen goede keuzen was geweest, omdat we hierdoor veel te veel verantwoordelijkheden in de klassen zetten. De keuzen om de RPG elementen uit de game weg te laten was wel een goede keuzen omdat we hierdoor wel een playable game konden opleveren.

Index

Inleiding

In dit project implementeren we de kennis die we opgedaan hebben tijdens de practica lessen door een game te bouwen. We zijn dit blok bezig geweest met het begrijpen van C++ na de beginselen van C afgelopen jaar. Nu brengen we die kennis samen door een game met behulp van C++ te schrijven. We hebben gebruik gemaakt van de SCRUM methode, een methodiek voor het werken in een team dat door middel van een iteratieve methode altijd een tussen product op hand heeft. Het doel van deze opdracht is om de kennis van C++ in praktijk te brengen en het werken in een team te bevorderen.

Het doel van dit verslag is om een technisch inzicht te geven van de bouw van de game. Dit verslag zal dieper ingaan op de ontwikkelingen binnen het proces, problemen bij de realisatie, een inzicht in het ontwerp van de game (game design) en de architectuur van de game.

Deze game is bedoeld voor jongeren, tieners en jongvolwassenen. Het zal deze doelgroep ook het meeste aanspreken.

In dit verslag zullen wij eerst het game design bespreken. Hiermee geven wij inzicht in de processen en keuzes die gemaakt werden om deze game in theorie tot stand te brengen. Hierna beschrijven wij de architectuur van het systeem door middel van een klassendiagram en de benodigde verklaringen daarbij. Hier geven wij ook aan wat we over de loop van de tijd verandert hebben. Verder kijken we naar de realisatie en de softwareontwikkeling, door naar alle processen te kijken die er voor zorgden dat de game tot stand kwam. Hier geven wij ook uitgebreidere beschrijving voor niet vanzelfsprekend delen in de code, problemen tijdens het ontwikkelen en wat voor tests we hebben uitgevoerd. We eindigen dit verslag met een evaluatie over hoe de samenwerking en ontwikkeling van ons project verliep en een conclusie met aanbevelingen op functioneel en technisch vlak.

Functioneel Ontwerp

Gate2158 is een 2D shooter waarin de speler van bovenaf het spel ziet. In dit spel moet u als speler met behulp van een pistool en een shotgun door meerdere levels vechten om aan het einde van het spel te komen. Dit kunt u doen door snel door het spel te rennen en alles te ontwijken of rustig elke vijand uit te schakelen om zo veilig door het level te komen.
De maps worden geladen door een maploader die bitmaps kan laden, deze kan u gemakkelijk zelf aanmaken in een tool zoals gimp, paint of paintdotnet. Door dit te doen kunt u ook zelf levels maken en die aan uw vrienden geven om te kijken of zij het level kunnen verslaan.
In het menu kunnen er maar een paar dingen worden gedaan, het spel starten en afsluiten.
De speler kan met WASD lopen, richten door de muis te bewegen en schieten met de linker muisknop. Deze controls zijn gekozen omdat deze universeel worden gebruikt.
Er is maar 1 soort wapen, een projectiel wapen. Dit wapen schieten 1 of meerdere projectielen met elk schot. Er zouden ook andere soorten wapens worden gemaakt, maar daar was geen tijd voor. De projectiel wapens kunnen makkelijk worden aangemaakt en erg verschillen van elkaar.

graphics stijl: Wij zijn voor een sci-fi look gegaan omdat dit goed bij het verhaal past, hiervoor hebben wij gratis art gevonden met een license waar wij geen problemen mee krijgen. Dit hebben wij gedaan omdat wij zelf geen teamleden hebben die game art kunnen maken.

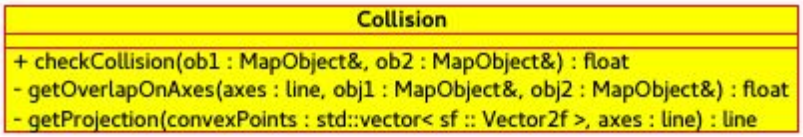
Wij waren eerst nog van plan om RPG elementen zoals een overworld, quests, npcs, towns enz toe te voegen maar dat zou te veel tijd kosten, in dat aspect waren wij iets te ambitieus.

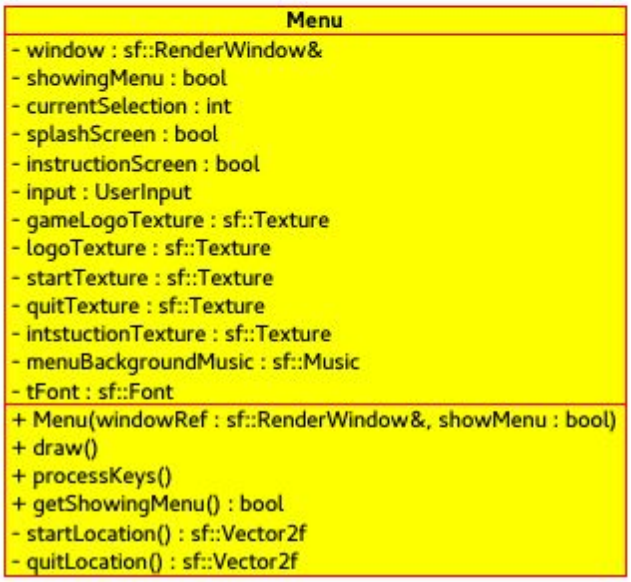
Technisch Ontwerp

In Gate2158 speelt het spel zich af in maps. Als een map voltooid is (de speler is bij de finish) wordt het volgende level geladen. De world-klasse omvat alle maps. De maps worden uit bitmaps ingeladen, waarbij de kleur van de pixel aangeeft wat er op die tile komt.

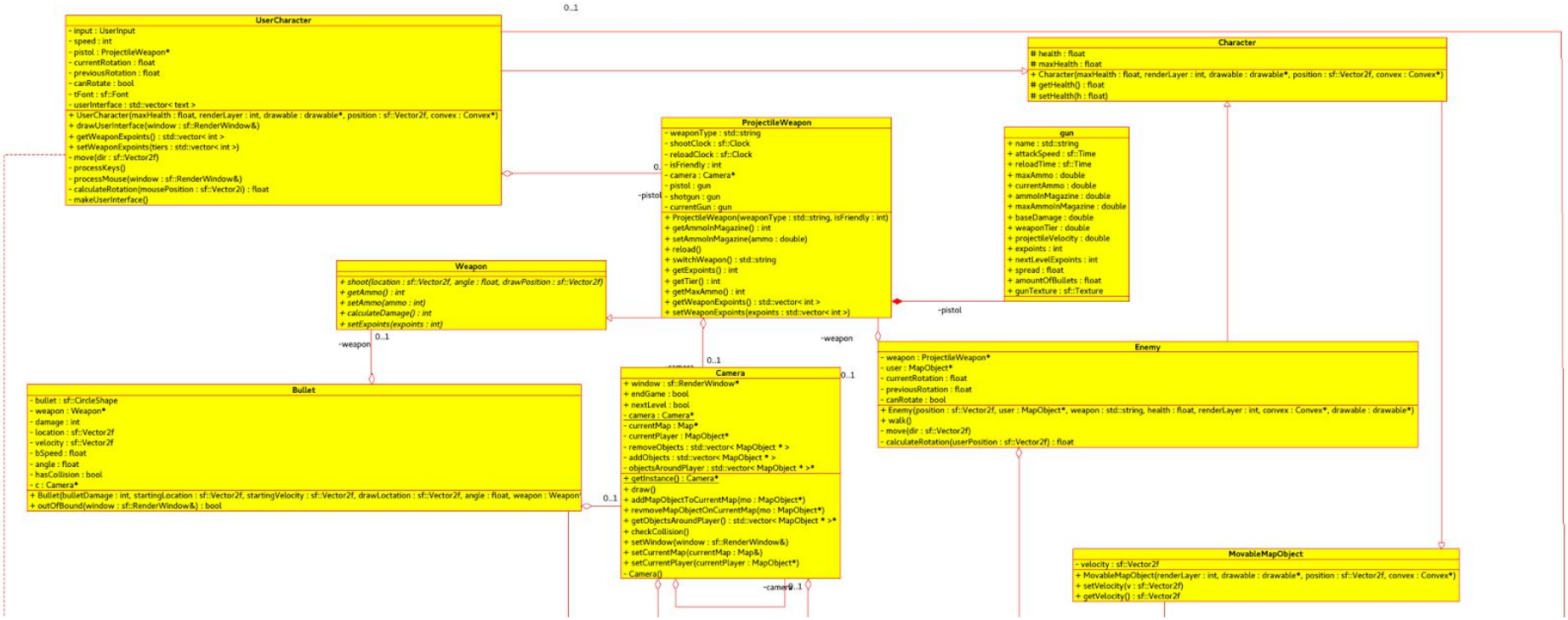
Alles wat op de kaart aanwezig is is indirect een MapObject. Zowel de speler als enemies stammen af van MovableMapObject. Door deze archicetcuur is bijvoorbeeld een collision engine eenvoudiger te bouwen omdat alle objecten waar de engine rekening mee moet houden een gemeenschappelijke interface hebben.

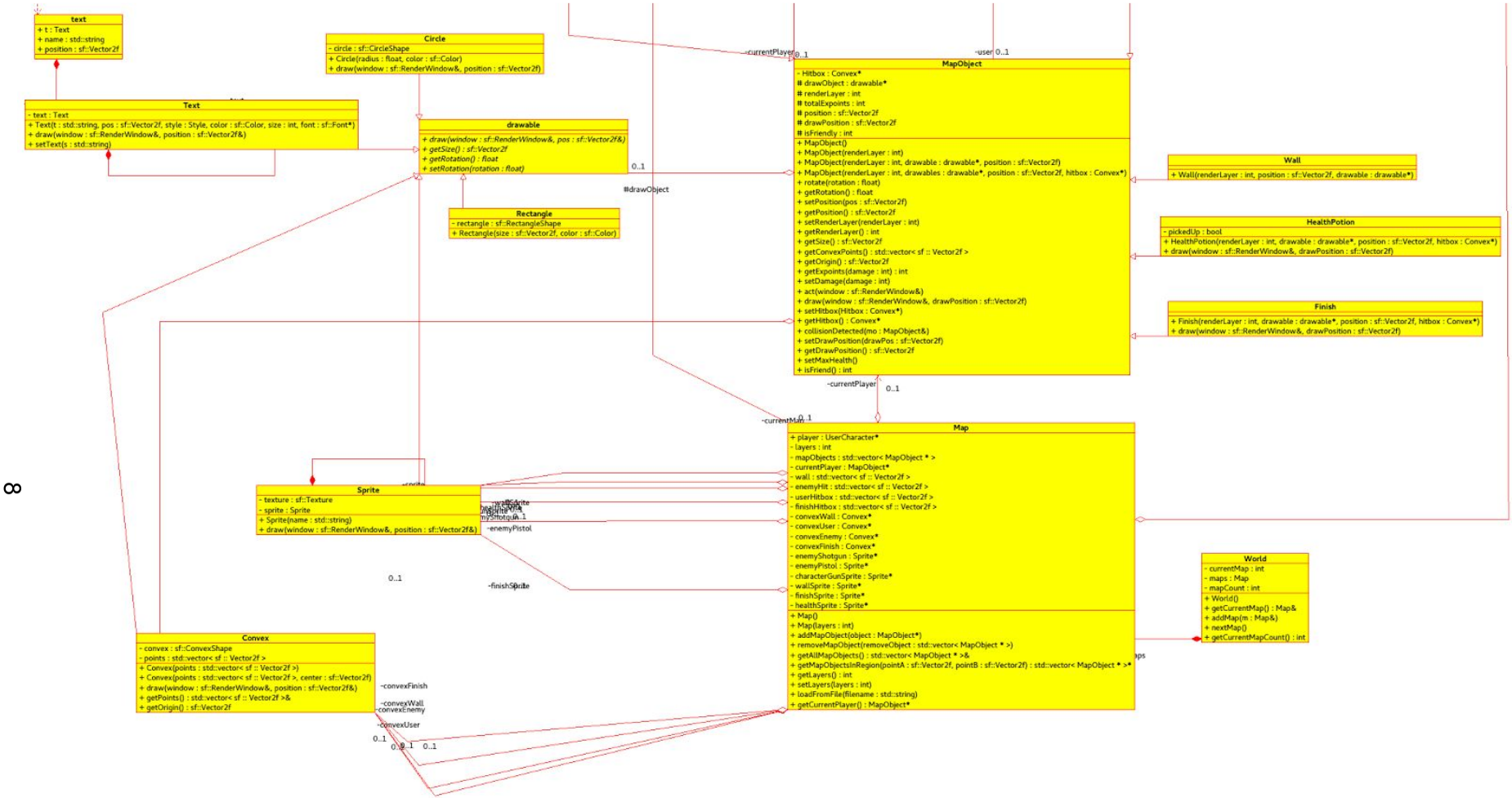
De camera klasse is verantwoordelijk voor het op de juiste plaats tekenen van de verschillende map objecten. Dit wordt gedaan door een offset bij de positie van het object op te tellen. De camera volgt de speler, waardoor de speler altijd in het midden van het scherm getekend wordt.





- Kernachtige beschrijving van de architectuur van het systeem
- In ieder geval het klassendiagram met motivering en beschrijving van de gemaakte keuzes
Zorg ervoor dat de modellen leesbaar zijn!


UserCharacter
- input : UserInput
- speed : int
- pistol : ProjectileWeapon*
- currentRotation : float
- previousRotation : float
- canRotate : bool
- tFont : sf::Font
- userInterface : std::vector< text >
+ UserCharacter(maxHealth : float, renderLayer : int, drawable : drawable*, position : sf::Vector2f, convex : Convex*)
+ drawUserInterface(window : sf::RenderWindow&)
+ getWeaponExpoints() : std::vector< int >
+ setWeaponExpoints(tiers : std::vector< int >)
- move(dir : sf::Vector2f)
- processKeys()
- processMouse(window : sf::RenderWindow&)
- calculateRotation(mousePosition : sf::Vector2i) : float
- makeUserInterface()
Character
health : float
maxHealth : float
+ Character(maxHealth : float, renderLayer : int, drawable : drawable*, position : sf::Vector2f, convex : Convex*)
# getHealth() : float
# setHealth(h : float)
ProjectileWeapon
- weaponType : std::string
- shootClock : sf::Clock
- reloadClock : sf::Clock
- isFriendly : int
- camera : Camera*
- pistol : gun
- shotgun : gun
- currentGun : gun
+ ProjectileWeapon(weaponType : std::string, isFriendly : int)
+ getAmmoInMagazine() : int
+ setAmmoInMagazine(ammo : double)
+ reload()
+ switchWeapon() : std::string
+ getExpoints() : int
+ getTier() : int
+ getMaxAmmo() : int
+ getWeaponExpoints() : std::vector< int >
+ setWeaponExpoints(expoints : std::vector< int >)
gun
+ name : std::string
+ attackSpeed : sf::Time
+ reloadTime : sf::Time
+ maxAmmo : double
+ currentAmmo : double
+ ammoInMagazine : double
+ maxAmmoInMagazine : double
+ baseDamage : double
+ weaponTier : double
+ projectileVelocity : double
+ expoints : int
+ nextLevelExpoints : int
+ spread : float
+ amountOfBullets : float
+ gunTexture : sf::Texture
Weapon
+ shoot(location : sf::Vector2f, angle : float, drawPosition : sf::Vector2f)
+ getAmmo() : int
+ setAmmo(ammo : int)
+ calculateDamage() : int
+ setExpoints(expoints : int)
Bullet
- bullet : sf::CircleShape
- weapon : Weapon*
- damage : int
- location : sf::Vector2f
- velocity : sf::Vector2f
- bSpeed : float
- angle : float
- hasCollision : bool
- c : Camera*
+ Bullet(bulletDamage : int, startingLocation : sf::Vector2f, startingVelocity : sf::Vector2f, drawLoctation : sf::Vector2f, angle : float, weapon : Weapon*)
+ outOfBound(window : sf::RenderWindow&) : bool
Camera
+ window : sf::RenderWindow*
+ endGame : bool
+ nextLevel : bool
- camera : Camera*
- currentMap : Map*
- currentPlayer : MapObject*
- removeObjects : std::vector< MapObject * >
- addObjects : std::vector< MapObject * >
- objectsAroundPlayer : std::vector< MapObject * >*
+ getInstance() : Camera*
+ draw()
+ addMapObjectToCurrentMap(mo : MapObject*)
+ revmoveMapObjectOnCurrentMap(mo : MapObject*)
+ getObjectsAroundPlayer() : std::vector< MapObject * >*
+ checkCollision()
+ setWindow(window : sf::RenderWindow&)
+ setCurrentMap(currentMap : Map&)
+ setCurrentPlayer(currentPlayer : MapObject*)
- Camera()
Enemy
- weapon : ProjectileWeapon*
- user : MapObject*
- currentRotation : float
- previousRotation : float
- canRotate : bool
+ Enemy(position : sf::Vector2f, user : MapObject*, weapon : std::string, health : float, renderLayer : int, convex : Convex*, drawable : drawable*)
+ walk()
- move(dir : sf::Vector2f)
- calculateRotation(userPosition : sf::Vector2f) : float
MovableMapObject
- velocity : sf::Vector2f
+ MovableMapObject(renderLayer : int, drawable : drawable*, position : sf::Vector2f, convex : Convex*)
+ setVelocity(v : sf::Vector2f)
+ getVelocity() : sf::Vector2f
0..1
-pistol
-weapon
-camera


text
+ t : Text
+ name : std::string
+ position : sf::Vector2f
Text
- text : Text
+ Text(t : std::string, pos : sf::Vector2f, style : Style, color : sf::Color, size : int, font : sf::Font*)
+ draw(window : sf::RenderWindow&, position : sf::Vector2f&)
+ setText(s : std::string)
Circle
- circle : sf::CircleShape
+ Circle(radius : float, color : sf::Color)
+ draw(window : sf::RenderWindow&, position : sf::Vector2f)
drawable
+ draw(window : sf::RenderWindow&, pos : sf::Vector2f&)
+ getSize() : sf::Vector2f
+ getRotation() : float
+ setRotation(rotation : float)
Rectangle
- rectangle : sf::RectangleShape
+ Rectangle(size : sf::Vector2f, color : sf::Color)
MapObject
- Hitbox : Convex*
drawObject : drawable*
renderLayer : int
totalExpoints : int
position : sf::Vector2f
drawPosition : sf::Vector2f
isFriendly : int
+ MapObject()
+ MapObject(renderLayer : int)
+ MapObject(renderLayer : int, drawable : drawable*, position : sf::Vector2f)
+ MapObject(renderLayer : int, drawables : drawable*, position : sf::Vector2f, hitbox : Convex*)
+ rotate(rotation : float)
+ getRotation() : float
+ setPosition(pos : sf::Vector2f)
+ getPosition() : sf::Vector2f
+ setRenderLayer(renderLayer : int)
+ getRenderLayer() : int
+ getSize() : sf::Vector2f
+ getConvexPoints() : std::vector< sf :: Vector2f >
+ getOrigin() : sf::Vector2f
+ getExpoints(damage : int) : int
+ setDamage(damage : int)
+ act(window : sf::RenderWindow&)
+ draw(window : sf::RenderWindow&, drawPosition : sf::Vector2f)
+ setHitbox(Hitbox : Convex*)
+ getHitbox() : Convex*
+ collisionDetected(mo : MapObject&)
+ setDrawPosition(drawPos : sf::Vector2f)
+ getDrawPosition() : sf::Vector2f
+ setMaxHealth()
+ isFriend() : int
Wall
+ Wall(renderLayer : int, position : sf::Vector2f, drawable : drawable*)
HealthPotion
- pickedUp : bool
+ HealthPotion(renderLayer : int, drawable : drawable*, position : sf::Vector2f, hitbox : Convex*)
+ draw(window : sf::RenderWindow&, drawPosition : sf::Vector2f)
Finish
+ Finish(renderLayer : int, drawable : drawable*, position : sf::Vector2f, hitbox : Convex*)
+ draw(window : sf::RenderWindow&, drawPosition : sf::Vector2f)
Map
+ player : UserCharacter*
- layers : int
- mapObjects : std::vector< MapObject * >
- currentPlayer : MapObject*
- wall : std::vector< sf :: Vector2f >
- enemyHit : std::vector< sf :: Vector2f >
- userHitbox : std::vector< sf :: Vector2f >
- finishHitbox : std::vector< sf :: Vector2f >
- convexWall : Convex*
- convexUser : Convex*
- convexEnemy : Convex*
- convexFinish : Convex*
- enemyShotgun : Sprite*
- enemyPistol : Sprite*
- characterGunSprite : Sprite*
- wallSprite : Sprite*
- finishSprite : Sprite*
- healthSprite : Sprite*
+ Map()
+ Map(layers : int)
+ addMapObject(object : MapObject*)
+ removeMapObject(removeObject : std::vector< MapObject * >)
+ getAllMapObjects() : std::vector< MapObject * >&
+ getMapObjectsInRegion(pointA : sf::Vector2f, pointB : sf::Vector2f) : std::vector< MapObject * >*
+ getLayers() : int
+ setLayers(layers : int)
+ loadFromFile(filename : std::string)
+ getCurrentPlayer() : MapObject*
World
- currentMap : int
- maps : Map
- mapCount : int
+ World()
+ getCurrentMap() : Map&
+ addMap(m : Map&)
+ nextMap()
+ getCurrentMapCount() : int
Sprite
- texture : sf::Texture
- sprite : Sprite
+ Sprite(name : std::string)
+ draw(window : sf::RenderWindow&, position : sf::Vector2f&)
Convex
- convex : sf::ConvexShape
- points : std::vector< sf :: Vector2f >
+ Convex(points : std::vector< sf :: Vector2f >)
+ Convex(points : std::vector< sf :: Vector2f >, center : sf::Vector2f)
+ draw(window : sf::RenderWindow&, position : sf::Vector2f&)
+ getPoints() : std::vector< sf :: Vector2f >&
+ getOrigin() : sf::Vector2f
-currentPlayer
0..1
-user
#drawObject
-currentMap
-enemyPistol
-finishSprite
-convexFinish
-convexWall
-convexEnemy
-convexUser

Realisatie

Tijdens het ontwikkelproces hebben we meerdere git branches gebruikt. Omdat we scrum als ontwikkelmethodiek gebruikte, hadden we de master branch die als eindproduct van iedere sprint functioneerde en een development branch om tussentijds branches samen te voegen. Verder werden voor de ontwikkeling van losse features branches vanuit development afgetakt, en na het afronden hiervan terug gemerged naar development. Hierdoor wordt voorkomen dat men in elkaars vaarwater gaat zitten.

We hebben er besloten, omdat onze groep bestaat naast windows gebruikers ook uit een linux gebruiker, dat wij de game zowel voor windows als linux ontwikkelen.

De installatie van de software liep niet volledig zonder problemen. Aangezien we verschillende operating systems in ons team hadden (Windows 10, Linux en oudere Windows versies) konden we niet allemaal met dezelfde Visual Basic versie werken. Dit bleek geen probleem te zijn, totdat we erachter kwamen dat je voor verschillende Visual Studio versies verschillende SFML versies nodig hebt. Door de verschillende versies op de remote repository te zetten kwam dit wel goed.

Naast de installatie problemen tussen de verschillende operating systems traden er ook wat problemen op rondom compatibiliteit tussen de Microsoft Visual C++ compiler en de GCC compiler van Linux. Uiteindelijk zijn deze problemen opgelost door, de stukken code waar de GCC compiler op vast liep, te herschrijven.

In onze game worden de kaarten opgeslagen als bitmaps. Iedere pixel in deze bitmaps stelt een tile van 32 bij 32 pixel voor. De kleur van de pixel bepaald wat voor object er geplaast wordt op het moment dat de kaart geladen wordt. Een zwarte pixel stelt bijvoorbeeld een muur voor en een rode pixel bepaald het spawn punt van de speler. De verschillende map objecten die in de maploader worden aangemaakt worden geallocceerd op de heap. De pointer hiervan er worden opgeslagen in een std::vector in de map.

We gebruiken de sf::Image klasse om de map in te laden. Op deze manier worden meerdere bestandsformaten ondersteund. Toch is belangrijk om een lossless bestandsformaat als PNG of BMP te gebruiken. Als een bestandsformaat met lossy compression, zoals JPEG, gebruikt wordt kunnen er namelijk artifacts ontstaan. Hierdoor kunnen pixels een andere kleur krijgen, die niet door de maploader worden herkend.

De camera van onze game, die verantwoordelijk is wat er op het scherm getekend wordt, is op een aparte manier geschreven. We hebben er namelijk een singleton klasse van gemaakt. Op deze manier konden we via de camera, de huidige map kunnen aanspreken. Dit hadden we nodig omdat de bullets en een enemies zichzelf van de map moeten kunnen

halen. Dit leek ons een beter alternatief dan de camera als parameter door te geven aan alle objecten die zichzelf verwijderen.

Tijdens het maken van de code, kwamen we een paar problemen tegen. Zo bleven enemies vast zitten in een muur, konden er op verschillende plekken niet geschoten worden en werd het spel werd erg sloom. Door met de debugger door het programma te gaan kwamen kwamen we er achter wat het probleem was.

Enemies bleven bijvoorbeeld vast zitten in de muur, omdat de collision de draw position gebruikte en deze niet correct werd bijgehouden. Door de draw position kon er ook niet op alle plekken geschoten worden, want deze werd bij het aanmaken van de bullet niet juist gezet. Hierdoor kwam de bullet het eerste frame op een plek waar die niet zou moeten zijn en als daar toevallig iets stond collide die met het object en verdween de bullet. Als er geen object in de weg stond werd de bullet het volgende frame netjes op de juiste plek getekend. Dit was erg moeilijk te achterhalen, maar met behulp van de debugger was het toch mogelijk.

Met behulp van ‘Performance and Diagnostics’ van Visual Studio, konden we achterhalen waarom het spel zo sloom werd. Het bleek aan de push back calls van de collision te liggen. Deze durden erg lang, en werden te vaak aangeroepen. We hebben uiteindelijk deze vervangen door een array en duplicaties uit de collision gehaald.

Evaluatie

Nadat de collision detection geschreven was, werd duidelijk dat deze wel werkte met de visual studio compiler van Windows, maar op Linux met de gcc compiler gaf het een foutmeldingen. We hadden een beter testproces moeten hebben voordat we de wijzigingen naar de development branch gemergeden.

Daarnaast hadden we, voordat we begonnen met de code te schrijven, beter moeten nadenken over het ontwerp. Toen we eenmaal bezig waren, kwamen we er namelijk achter dat we veel fouten hadden gemaakt in het ontwerp. Hierdoor moesten we problemen oplossen, die vermeden hadden kunnen worden door van tevoren goed na te denken over het ontwerp. We hadden beter, toen het nog niet te laat was, een verbetering in het ontwerp kunnen maken. Hierdoor zouden we later tijd besparen.

We hadden iets eerder moeten beginnen aan de camera klassen. Toen we de implementatie van de camera hadden geschreven, kwamen er namelijk veel problemen in andere klasse. Zo werkten de collision niet meer en werden bullets verkeerd getekend.

Het werken in een team ging over het algemeen wel goed. We hielden veel contact met elkaar door middel van de chat service Telegram en iedereen deed zijn deel in het project goed. Ook geen problemen met ongemelde afwezigheid.

Conclusies en aanbevelingen

Gate 2158 is een spel dat bij een groot publiek zal aanspreken. Het combineert de wel bekende top-down style met moderne shooter elementen. Door de originele map loading engine die maps genereert vanuit een bitmap kan iedereen makkelijk hun eigen level samen stellen en spelen. Het is ook makkelijk deze levels te delen met andere mensen aangezien het gewoon een plaatje is. Ook is de speler niet geforceerd een bepaald pad te nemen. De speler kan ervoor kiezen zoveel mogelijk monsters te doden om wapens te upgraden, of zo snel mogelijk door het level heen proberen te gaan.

Functioneel gezien, hebben we niet alles wat we wilde hebben ook echt gerealiseerd. De basis van de game hebben we wel gemaakt, maar bijvoorbeeld het RPG element konden we, door tijd gebrek, niet implementeren. Daarnaast kunnen er nog veel meer levels en wapens gemaakt worden.

Op technisch vlak hebben we de klassen goed gecodeerd en het werkt goed samen. Alleen is de klasse architectuur is niet helemaal goed gevormd. De verantwoordelijkheden van de klassen zijn daardoor niet goed afgesplitst, en is het moeilijk om nieuwe functionaliteiten toe te voegen. Daar zijn dus nog wel verbeteringen op aan te brengen.

Bijlagen