‘Technisch ontwerp T^NKS!

Versie 2.0

22 MEI

Klas: ITA 1DD
Gemaakt door: Osama Halabi en Robert Boudewijn
Studenten nummer: 628160 en 631286
Docent: ir. B. van der Wal

HAN_UNIVERSITY OF APPLIED SCIENCES

Inhoudsopgave

1 Inleiding

Voor het vak OOPD van onze opleiding moeten wij een spel maken voor afronding van het vak. In dit document behandelen wij alle technische onderdelen voor het bouwen van dit spel.

Het spel heet ‘T^NKS!’ en is gebaseerd op het WII Play spel Tanks. (Onbekend, 2019)

Het doel van het spel is om in elk level alle andere tanks uit te schakelen door ze te beschieten met kogels of door mijnen te plaatsen. Het spel speelt zich af in een level ter grote van het speelscherm. Met verschillende obstakels.

2. Game engine

De game-engine die wij moeten gebruiken voor ons beroepsproduct van OOPD (de game). De engine werd in 2014-2015 ontwikkeld door een aantal OOSE-studenten en wordt momenteel doorontwikkeld door docenten en studenten van ICA.

De game-engine is op github te vinden en daarop staat de wiki ervan waarmee de game-engine beter kunnen gebruiken en begrijpen. Ook is de Javadoc daar te vinden. De Javadoc is een erg handige tool om de verschillende functies en methodes te kunne begrijpen.

3. Eigen classes

Voor het beroepsproduct moesten we minimaal 8 klassen maken maar voor het spel hebben we 13 klassen gemaakt. Hieronder leggen we de klassen uit en in de bijlage is de klasdiagram te vinden met de naam: classdiagram.

De klassen:

3.1 Game

De klasse game is de main klasse van ons spel en hij extends de GameEngine. Waarin de setup, speler, Enemy en de blocken worden getekend.

De klas Game heeft meerder methoden en ik leg alle methoen hieronder uit:

setupGame():
Hier in geven we de grote van het beelde van het spel daarna teken we de tiels, de dasboard daarna de takns.

initializeTileMap():
Hierin maken maken we de tile map dus waar moeten de tile plaats vind in het beeld.

dashboardBuild():
Hierin wordt de dashboard gemaakt.

dashboardUpdate():
Door deze methode kunnen we gemakkelijk de dashboard updaten.

buildFirstLevelBreakableBlocks():
Door deze methode teken we de tanks voor de eerste level van het spel.

placeTankPlayer(int id, float xPos, float yPos):
Hierin maken we een nieuw game object met de naam tank player en we geven hem de id, x posietie en y posietie mee.

placeTankEnemy(int id, float xPos, float yPos, int type):

Hierin maken we een nieuw game object met de naam tank enemy en we geven hem de id, x posietie en y posietie mee.

placeTanksFirstLevel():
hierin maken we gebruie van de methoden placeTankPlayer() en placeTankEnemy():

3.2 Tank

Tank is de parent klasse van alle tanks.
Hij heeft de volgende Methodes:

placeRocketSlow(int id, float xPos, float yPos, float direction, int amountOfBounce):
Deze methode zet rocketSlows neer. En geeft ze de hoek waar ze naar toe moeten.

placeRocketFast(int id, float xPos, float yPos, float direction):
Deze methode zet rocketFasts neer. En geeft ze de hoek waar ze naar toe moeten.

tankShootRocketSLow(float angle):
Deze methode zet rocketSlows neer op de positie van de tank en heeft alleen de hoek nodig om te werken.

abstract tileCollisionOccurred(List<CollidedTile> collidedTiles):
Dit is een abstracte methode zodat er in de child klassen deze kan worden overschreven.

3.2.1 TankEnemy

TankEnemy is de child van Tank en de parent van de specifieke tanks. Hij doet niet erg veel behalve parent spelen en een abstracte methode introduceren.

abstract driveAndAttackPlayer():
Dit is een abstracte methode zodat polymorfie kan worden toegepast.

3.2.1.1 TankEnemyGray

Dit is de klasse van alle grijze vijanden. Hierin wordt zijn beweging en doen beschreven. Dit werkt door de volgende methodes:
driveAndAttackPlayer():
Dit is een klasse die wordt overgerfd van de TankEnemy klasse. En zorgt er voor dat de tank zelf rijdt en na denkt. Eerst gaat de tank rondjes rijden en zoeken naar de player. Wanneer hij deze heeft gevonden dan kan hij dichter bij komen en daarna gaan aanvallen.

3.2.1.2 TankEnemyBrown

TankEnemyBrown is erg het zelfde als TankEnemyGray. Alleen kan de bruine tank niet rijden en daarom is zijn DriveAndAttackPlayer anders. In die methode wordt alleen gekeken naar de speler en wanneer de speler in de buurt is zal de tank pas gaan aanvallen.

3.2.2 TankPlayer

TankPlayer is eigenlijk de Speler. Deze klasse heeft meerdere methodes:

mouseMoved(int x, int y):
Deze methode houdt bij waar de muis van de speler zich bevind op het scherm.

keyPressed(int keyCode, char key):
De methode KeyPressed is een methode die zorgt voor de beweging van de speler. Wanneer je op W drukt zal je gaan bewegen naar voren etc.

getLifes():
GetLifes() is een getter voor de hoeveelheid overige leven van de speler.

setLifes(int lifes):
SetLifes is een setter voor de hoeveelheid overige leven van de speler.

gameObjectCollisionOccurred(List<GameObject> collidedGameObjects):
Deze methode ceckt of de tank geraakt wordt door een kogel of door een andere tank. Wanneer dit gebeurd start het level opnieuw en gaat er een leven van af. Wanneer er geen levens meer zijn dan stopt het spel en komt de spelerTank niet terug.

3.3 Ammo

Ammo is de parent van de klas AmmoRocket en hij is abstract klas die extends SpriteObject van de game-engine en implementeert de *ICollidableWithTiles in deze klas override we de (tileCollisionOccurred(List<CollidedTile> collidedTiles)) van de ICollidableWithTiles klas.*

3.3.1 AmmoRocket

AmmoRocket is de child van Ammo en hij implementeert de klas *ICollidableWithGameObjects* van de game-engine en met Ammo overriden we de methodes (tileCollisionOccurred(List<CollidedTile> collidedTiles) en gameObjectCollisionOccurred(List<GameObject> collidedGameObjects)) van de *ICollidableWithGameObjects en de ICollidableWithTiles.*

3.3.1.1 AmmoRocketSlow

AmmoRocketSlow is de klasse waarin de slomere raket beschreven staat. Het heeft de volgende methodes:

tileCollisionOccurred(List<CollidedTile> collidedTiles):
Hiermee check je of de raket een muur raakt en wat hij dan moet doen. In dit geval checken of hij nog een keer kan stuiteren en daarna zichzelf verwijderen.

gameObjectCollisionOccurred(List<GameObject> collidedGameObjects):
Deze methode checkt of hij een ander oject raakt en verwijderd dan dat object. Waardoor terugkatsing of het raken van andere tanks direct dat object verwijderd.

placeRocketSlow(int id, float xPos, float yPos, float direction, int amountOfBounce)
Hier is een precies de zelde placeRocketSlow als in Tank. Echter is deze in deze klasse omdat het er voor zorgt dat het een ricochet toelaat.

3.3.1.2 AmmoRocketFast

AmmoRocketSlow is de klasse waarin de snellere raket beschreven staat. Het heeft de volgende methodes:

tileCollisionOccurred(List<CollidedTile> collidedTiles):

Hiermee check je of de raket een muur raakt en wat hij dan moet doen. In dit geval zich zelf verwijderen.

gameObjectCollisionOccurred(List<GameObject> collidedGameObjects):
deze methode checkt of hij een ander oject raakt en verwijderd dan dat object.
Waardoor terugkatsing of het raken van andere tanks direct dat object verwijderd.

3.4. Block

Block is de parent van 'BlockBreackable' Origineel wilden we alle 3 de type blokken hieronder laten vallen. Later hebben we besloten dat we alleen nog blockbreackable wilden houden. Daarom heeft Block ook alleen maar een constructor.

3.4.1 BlockBreackable

BlockBreackable (hierna BBable) is een klasse die we niet gebruiken maar wel deels uitgeprogrammeerd is. Eerst wilde we dat deze klasse door mijnen kapot kan worden gemaakt. Echter hebben we nu besloten dat het ook kan door objecten van 'AmmoRocketFast' kan. BBable is ook geen tile. Zoals de andere blokken. BBable is een gameobject. Omdat we willen dat BBable kapot kan gaan. Zoals de naam ook suggereert. De volgende methodes heeft de klasse waarvan de meeste outdated zijn:

isBroken() :
Outdated
Is een getter voor broken.

setBroken():
Outdated
Is een setter voor broken.

getAmount() :
Outdated
Is een getter voor Amount.

setBroken():
Outdated
Is een setter voor Amount.

4. Polymorfie

Voor dit product willen we graag polymorfie toepassen. Polymorfie is het overerven van methodes van ‘parent’ klassen. En deze overschrijven. Waardoor de ‘parent’ klassen veelvormigheid krijgen.

4.1 Zelf toegepast

Wij willen zelf dus polymorfie toepassen. Meerdere keren zelfs. De belangrijkste toepassing gaat het rijden en attacken zijn van de enemy tank. Elke tank moet dit anders gaan aanpakken en daardoor wordt de klasse ‘EnemyTank’ veelvormig.

4.2 De gebruikte methode:

```
@Override
public void driveAndAttackPlayer() {};
```

Wij hebben hiervoor gekozen omdat dit ons de beste mogelijkheid lijkt om polymorfie toe te passen.
We willen ook polymorfie gaan toepassen bij ‘Ammo’. En specifiek gezien ‘AmmoRocket’ Omdat er meerdere rakketen gaan zijn die een andere functie hebben.

5 Conclusie

We vonden dit beroeps product best lastig. De ongeplande veranderingen en het leren van object georiënteerd Java is natuurlijk niet makkelijk maar dit project heeft ons wel helpen begrijpen hoe het precies werkt. Het gene dat we het meest lastigste vonden was het vooraf bedenken wat we gingen coderen.

Figuren

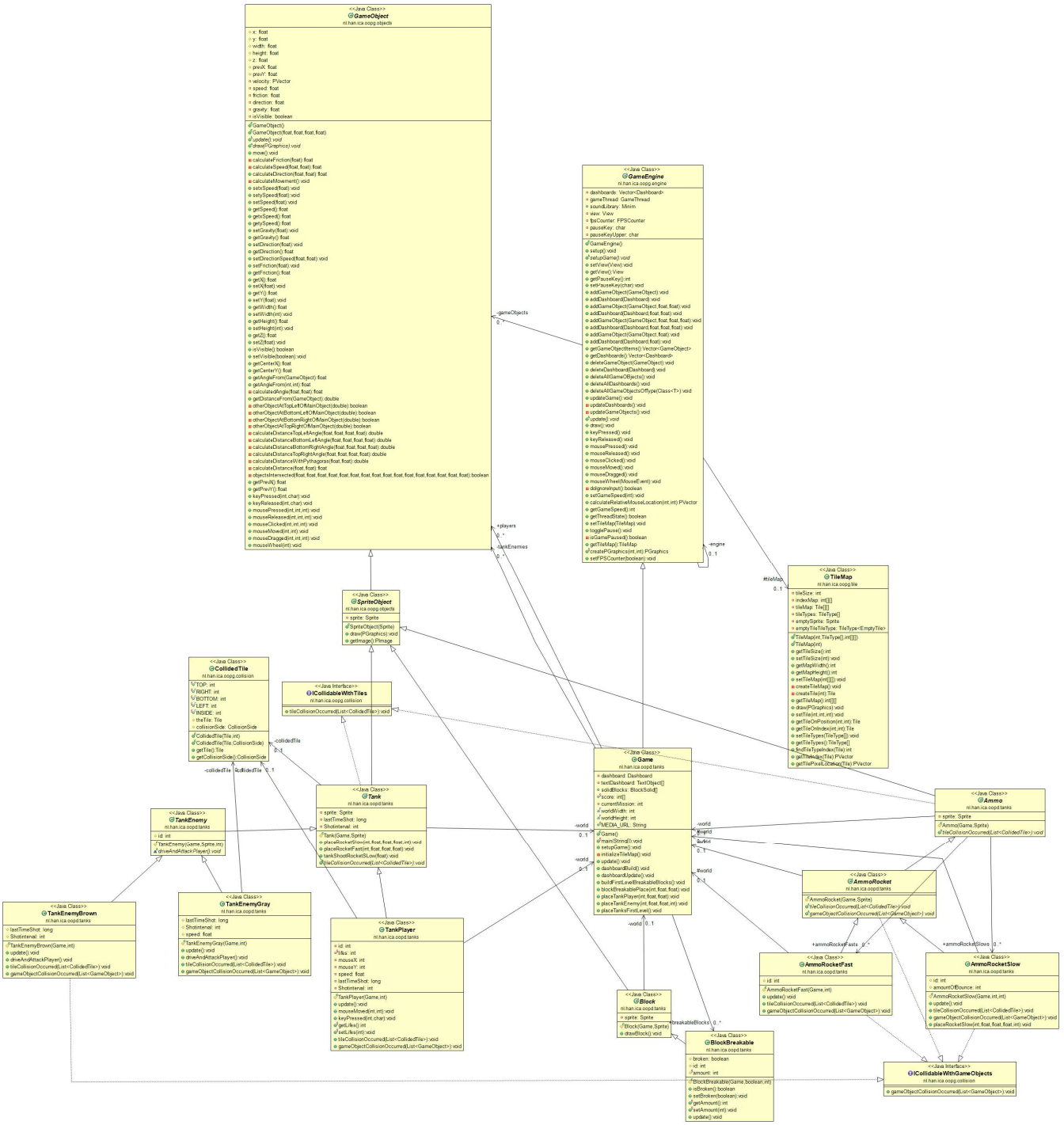
Bronnen

Onbekend. (2019, november 21). *Tanks!* Opgehaald van Nintendo.fandom: https://nintendo.fandom.com/wiki/Tanks! op 18/03/2020

projectmanagementsite. (2019, onbekend onbekend). *MoSCoW*. Opgehaald van projectmanagementsite: https://projectmanagementsite.nl/moscow/#.XnHp-KhKiMo op 18/03/2020

Bijlagen:

A1 Classdiagram.


GameObject
GameEngine
TileMap
SpriteObject
Ammo
CollidableTile
ICollidableWithTiles
Tank
Game
TankEnemy
TankEnemyBrown
TankEnemyGray
TankPlayer
AmmoRocket
AmmoRocketFast
AmmoRocketSlow
Block
BlockBreakable
ICollidableWithGameObjects