Rapport de projet CPS

Spécification formelle et implémentation du jeu Dungeon Master

Paul Charton

29 Mai 2018

Table des matières

Introduction

Lors de ce rapport, je vais vous présenter toutes les spécifications et implémentaion que j'ai effectué tout au long de ce projet. J'expliquerais avec un détails particulier les spécification que j'ai moi-même ajoute a ce jeu et à l'ensemble de spécification déjà existante.
Je vais commencer lors de cette introduction a lister les éléments que j'ai ajouter, vous les retrouverez en détails plus tard dans ce rapport.
Malheuresement suite a un manque de temps et mauvaise organisation, le rapport que vous allez lire, la spécification associé ainsi que l'implementation peuvent être incomplete et comporter des erreurs.

Chapitre 1

Ma contribution

1.1 "Chest" et son contenu

L'une de mes premieres volonté à été d'ajouter un service "Chest", cela pour renforcer le l'aspect "Porte-Monstre-Trésor" que je voulais donné à ce jeu. Le service étant le plus proche de ce concept a été "Mob", mais il ne me convenait pas car ceux-ci peuvent bouger. J'ai donc commencé par crée un service "MotionlessObject" qui reprend les même caractéristique que celle de "Mob" mais en enlevant la possibilité de bouger l'objet. J'en ai également profiter pour faire hériter "Mob" de ce service. On a maintenant un objet immobile a la base. Puis on rend cette objet mobile a l'aide de "Mob". Et enfin on ajoute la vie avec "Entity". Tout cela avec des lien d'héritage. J'ai ensuite rajouter les méthodes "Loot" et "Destroy" à "MotionlessObject". En symmetrie j'ai ajouter les méthode "takeObject" et "attack" réspectivement aux service "Player" et aux service "Entity", pour permettre a ces objet d'intéragir. "takeObject" n'aura d'effet que sur "Chest" et "MotionlessObject", cela activera leur méthode "Loot". Cette méthode "Loot" aura pour effet de choisir un service "LootObject" parmi une liste et de l'ajouter dans l'inventaire du joueur à l'aide de la méthode "addLoot".
La méthode "attack" quand à elle aura un effet sur tout ce groupe de service. Sur le service "MotionlessObject", "attack" déclenchera directement la méthode "destroy", de même pour le service "Mob". Par contre à partir de entity, la méthode "attack" déclenchera la méthode "getDamage", qui va permettre de faire diminuer les points de vie de l'objet qui se fait attaquer.

1.2 Un nouveau monstre

La création du service "Chest" est lié a une autre envie. Celle de re-crée le Mimic de Dark Souls. En effet, ce monstre est différent des autres de par sa manieres d'agir. Au lieu de courir aprés le Joueur, il va simplement l'attendre. Cela en fait donc un monstre immobile. C'est pour cette raison que je le fais hériter directement du service "MotionlessObject". A la difference que quand le Joueur appelera la méthode "loot" d'un mimic, il aura une petite surprise.

1.3 Les potions

Encore un concept que j'ai repris de Dark Souls. Tout au long de l'aventure, soit en ouvrant des coffres, soit en tuant des mimics, on peut trouver des potions. Mais plus que de simple comsomable, ce sont des contenant, que l'on boire et remplir a volonté.

1.4 La taille

J'ai voulu, dans ma version de Dungeon Master, donner une importance a la taille du joueur par rapport a celle des monstres. En effet, le joueur va pouvoir équiper trois pieces d'armures différentes ; un casque, un torse et des jambieres. Le rapport de taille entre le montre et le joueur décidera de quelle sera la piece d'armure sera comtabiliser dans la réduction des dégats. Il est toutefois important de bien s'équiper, toutes les pieces d'armures augmentant les points de vies maximum.

1.5 Ce que j'aurais voulu faire

Avec plus de temps j'aurais implementer ou fini d'implementer d'autre idées parmi celles-ci :

Ambidextrie : On peut taper les trois cases devant soi, mais les dégats dépendent de l'arme de la main associé.

Fontaine de Vie : Principalement pour remplir les potions.

Porte invisible : Simplement des racourcie avec potentiellement trésors ou mimic bonus.

Chapitre 2

Spécification formelle

2.1 Les type utilisée

type Cell {**IN**, **OUT**, **EMP**, **WLL**, **DNO**, **DNC**, **DWO**, **DWC** }
type Dir {**N**, **S**, **W**, **E** }
type Command {**FF**, **BB**, **RR**, **LL**, **TL**, **TR**, **AA**, **TK**}
type ArmorSet {**MIMIC**, **CHEST**, **SKELETON** }
type Set[T]
type Array[T1,...,TN]

2.2 LootObjectService

Service : LootObject

2.3 ArmorPieceService

Service : ArmorPiece **includes** LootObject
Types : String, ArmorSet, int
Observator : **const** `getName` : [ArmorPiece] → String
const `getArmoSet` : [ArmorPiece] → ArmorSet
const `getArmor` : [ArmorPiece] → int
const `getHP` : [ArmorPiece] → int
Constructors : `init` : String · ArmorSet · int · int → [ArmorPiece]
pre `init(n,s,a,hp)` **requires** $0 \leq$ a **and** $0 \leq$ hp
Observation :
[invariant] : $\top$
[init] : `getName(init(n,s,a,hp))` = `n`
`getArmorSet(init(n,s,a,hp))` = `s`
`getArmor(init(n,s,a,hp))` = `a`
`getHP(init(n,s,a,hp))` = `hp`

2.4 WeaponService

Service : Weapon **includes** LootObject
Types : String, ArmorSet, int
Observator : **const** `getName` : [Weapon] → String
const `getArmoSet` : [Weapon] → ArmorSet
const `getArmor` : [Weapon] → int
const `getHP` : [Weapon] → int
Constructors : `init` : String · ArmorSet · int · int → [Weapon]
pre `init(n,s,a,hp)` **requires** $0 \leq$ a **and** $0 \leq$ hp
Observation :
[invariant] : $\top$
[init] : `getName(init(n,s,a,hp))` = `n`
`getArmorSet(init(n,s,a,hp))` = `s`
`getArmor(init(n,s,a,hp))` = `a`
`getHP(init(n,s,a,hp))` = `hp`

2.5 PotionService

Service :	Potion **includes** LootObject
Types :	boolean
Observator :	`isFull` : [Potion] → boolean
Constructors :	`init` : void → [Potion]
Operators :	`getDrinked` : [Potion] → [Potion]
	pre `getDrinked(P)` **requires** `isFull(p)` = ⊤
	`getFilled` : [Potion] → [Potion]
Observation :	
[invariant] :	isFull(getFilled(P)) = ⊤
[init] :	isFull(init()) = ⊥ [getDrinked] : isFull(getDrinked(P)) = ⊥
[getFilled] :	isFull(P) = ⊤ **implies** isFull(getFilled(P)) = ⊤
	isFull(P) = ⊥ **implies** isFull(getFilled(P)) = ⊤

2.6 MotionlessObjectService

Service : MotionlessObject
Types : Environment, int, Dir
Observator : `getEnvironment` : [MotionlessObject] → Environment
const `getCol` : [MotionlessObject] → int
const `getRow` : [MotionlessObjectt] → int
const `getDir` : [MotionlessObject] → Dir
Constructors : `init` : Environment · int · int · Dir → [MotionlessObject]
pre `init(E,x,y,D)` **requires** $0 \leq$ `x` $<$ `Environment::Width(E)`
and $0 \leq$ `y` $<$ `Environment::Height(E)`
Operators : `Loot` : [MotionlessObject] → [MotionlessObject]
pre getDir(M) = N **implies**
getCellContent(getEnvironment(M), getRow(M) + 1, getCol(M))
= PlayerService **and** getDir(P) = S
pre getDir(M) = S **implies**
getCellContent(getEnvironment(M), getRow(M) - 1, getCol(M))
= PlayerService **and** getDir(P) = N
pre getDir(M) = E **implies**
getCellContent(getEnvironment(M), getRow(M), getCol(M) + 1)
= PlayerService **and** getDir(P) = W
pre getDir(M) = W **implies**
getCellContent(getEnvironment(M), getRow(M), getCol(M) - 1)
= PlayerService **and** getDir(P) = E

`Destroy` : [MotionlessObject] → [MotionlessObject]
pre getDir(M) = N **implies**
getCellContent(getEnvironment(M), getRow(M) + 1, getCol(M))
= PlayerService **and** getDir(P) = S
pre getDir(M) = S **implies**
getCellContent(getEnvironment(M), getRow(M) - 1, getCol(M))
= PlayerService **and** getDir(P) = N
pre getDir(M) = E **implies**
getCellContent(getEnvironment(M), getRow(M), getCol(M) + 1)
= PlayerService **and** getDir(P) = W
pre getDir(M) = W **implies**
getCellContent(getEnvironment(M), getRow(M), getCol(M) - 1)
= PlayerService **and** getDir(P) = E

Observation :

[invariant] : $0 \leq$ `Col(M)` $<$ `Environment::Width(Envi(M))`
$0 \leq$ `Row(M)` $<$ `Environment::Height(Envi(M))`
Environment : :CellNature(Envi(M),Col(M),Row(M)) $\notin$
{**WLL**, **DNC**, **DWC**}

[init] : `Col(init(E,x,y,D))` = `x`
`Row(init(E,x,y,D))` = `y`
`Face(init(E,x,y,D))` = `D`
`Envi(init(E,x,y,D))` = `E`

2.7 MimicService

Service : Mimic **includes** MotionlessObject

Observator : `getHP` : [Mimic] $\rightarrow$ int
const `getHeight` : [Mimic] $\rightarrow$ int
`isAwake` : [Mimic] $\rightarrow$ boolean

Constructors : `init` : Environment $\cdot$ int $\cdot$ int $\cdot$ Dir $\rightarrow$ [Mimic]
pre `init(E,x,y,D)` **requires** $0 \leq$ `x` $<$ `Environment::Width(E)`
and $0 \leq$ `y` $<$ `Environment::Height(E)`

Operators : `attack` : [Mimic] $\rightarrow$ [Mimic]
`getDamage` : [Mimic] $\cdot$ int $\rightarrow$ [Mimic]
pre getHP(M) $>$ 0 **Observation** :

[invariant] : attack(M) **implies** isAwake(M) = $\top$

[init] : isAwake(init(E,x,y,D)) = $\bot$

[getDamage] : getHP(getDamage(M,x)) = getHP(M) - x

2.8 MobService

Service : Mob **includes** MotionlessObject

Operators :
`forward` : [Mob] → [Mob]
`backward` : [Mob] → [Mob]
`turnLeft` : [Mob] → [Mob]
`turnRight` : [Mob] → [Mob]
`strafeLeft` : [Mob] → [Mob]
`strafeRight` : [Mob] → [Mob]

Observations :

[invariant] :
$0 \leq$ `Col(M)` $<$ `Environment::Width(Envi(M))`
$0 \leq$ `Row(M)` $<$ `Environment::Height(Envi(M))`
Environment : :CellNature(Envi(M),Col(M),Row(M)) $\notin$
{**WLL**, **DNC**, **DWC**}

[Forward] :
`Face(M)`=**N implies**
`Environment::CellNature(Envi(M),Col(M),Row(M)+1)` $\in$ {**EMP**, **DWO**}
and `Row(M)+1` $<$ `Environment::Width(Envi(M))`
and `Environment::CellContent(Envi(M),Col(M),Row(M)+1)` = **No**
implies `Row(Forward(M))` = `Row(M) + 1`
and `Col(Forward(M))` = `Col(M)`
`Face(M)`=**N implies**
`Environment::CellNature(Envi(M),Col(M),Row(M)+1)` $\notin$ {**EMP**, **DWO**}
or `Row(M)+1` $\geq$ `Environment::Width(Envi(M))`
or `Environment::CellContent(Envi(M),Col(M),Row(M)+1)` $\neq$ **No**
implies `Row(Forward(M))` = `Row(M)`
and `Col(Forward(M))` = `Col(M)`
`Face(M)`=**E implies**
`Environment::CellNature(Envi(M),Col(M)+1,Row(M))` $\in$ {**EMP**, **DNO**}
and `Col(M)+1` $<$ `Environment::Height(Envi(M))`
and `Environment::CellContent(Envi(M),Col(M)+1,Row(M))` = **No**
implies `Row(Forward(M))` = `Row(M)`
and `Col(Forward(M))` = `Col(M) + 1`
`Face(M)`=**E implies**
`Environment::CellNature(Envi(M),Col(M)+1,Row(M))` $\notin$ {**EMP**, **DWO**}
or `Row(M)` $\geq$ `Environment::Width(Envi(M))`
or `Environment::CellContent(Envi(M),Col(M)+1,Row(M))` $\neq$ **No**
implies `Row(Forward(M))` = `Row(M)`
and `Col(Forward(M))` = `Col(M)`

Face(M)=**S implies**
Environment::CellNature(Envi(M),Col(M),Row(M)-1) $\in$ {**EMP**, **DWO**}
and Col(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = **No**
implies Row(Forward(M)) = Row(M) - 1
and Col(Forward(M)) = Col(M)
Face(M)=**S implies**
Environment::CellNature(Envi(M),Col(M),Row(M)-1) $\notin$ {**EMP**, **DWO**}
or Col(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ **No**
implies Row(Forward(M)) = Row(M)
and Col(Forward(M)) = Col(M)
Face(M)=**W implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in$ {**EMP**, **DNO**}
and Row(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = **No**
implies Row(Forward(M)) = Row(M)
and Col(Forward(M)) = Col(M) - 1
Face(M)=**W implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\notin$ {**EMP**, **DNO**}
or Row(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ **No**
implies Row(Forward(M)) = Row(M)
and Col(Forward(M)) = Col(M)

de même pour les autres fonctions.

2.9 EntityService

Service : Entity **includes** Mob

Observator : `getHP` : [Entity] → int

const `getHeight` : [Entity] → int

Constructors : `init` : Environment · int · int · Dir · int · int → [Entity]

pre `init(E,x,y,D, hp, h)` **requires** $0 \leq$ `x` $<$ `Environment::Width(E)`

and $0 \leq$ `y` $<$ `Environment::Height(E)`

and hp > 0

and h > 0

Operators : `attack` : [Entity] → [Entity]

`getDamage` : [Entity] · int → [Entity]

pre getHP(E) > 0

`step` : [Entity] → [Entity]

Observation :

[init] : getHP(init(E,x,y,D,hp,h)) = hp

getHeight(init(E,x,y,D,hp,h)) = h

[getDamage] : getHP(getDamage(E,x)) = getHP(E) - x

2.10 Player

Service : Player **includes** Entity

Observator :
`LastCom` : [Player] → Option[Command]
`Content` : [Player] · int · int → Option[Mob]
pre `Content(P,x,y)` **requires** x ∈ {-1,0,1}**and** y ∈ {-1,+3}
`Nature` : [Player] · int · int → Cell
pre `Nature(P,x,y)` **requires** x ∈ {-1,0,1}**and** y ∈ {-1,+3}
`Viewable` : [Player] · int · int → Cell
pre `Nature(P,x,y)` **requires** x ∈ {-1,0,1}**and** y ∈ {-1,+3}
`getNbPotion` [Player] → int
`getHelm` [Player] → ArmorPiece
`getChest` [Player] → ArmorPiece
`getLeg` [Player] → ArmorPiece
`getRightHand` [Player] → Weapon
`getLeftHand` [Player] → Weapon

Operator
`setLastCommand` [Player] · Command → [Player]
`addLoot` [Player] · LootObject → [Player]
`takeObject` [Player] → [Player]
`drinkPotion` [Player] → [Player]
`equipHelm` [Player] · Helms → [Player]
`equipChest` [Player] · ChestArmor → [Player]
`equipHelm` [Player] · LegArmor → [Player]
`equipRightHand` [Player] · Weapon → [Player]
`equipLeftHand` [Player] · Weapon → [Player]
`unequipHelm` [Player] → [Player]
`unequipChest` [Player] → [Player]
`unequipLeg` [Player] → [Player]
`unequipRightHand` [Player] → [Player]
`unequipLeftHand` [Player] → [Player]