ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

Projet Zorglub

Charlet Justine, Descatoire Théau
Montmaur Antoine, Sathiakumar Kuga

6 Octobre 2020

Table des matières

Chapitre 1

Environnement de développement

1.1 Introduction

Le Projet Logiciel Transverse (PLT) est un projet de 120h destiné à la conception d'un jeu vidéo de type tour par tour, avec un réseau multi joueur en ligne. Il fait intervenir la plupart des cours dispensés en Informatique et Systèmes à l'ENSEA : génie logiciel, algorithmique, programmation parallèle et web services.

Au cours de ce projet, nous allons faire de la conception UML, réaliser un moteur d'affichage, créer une IA basique, une heuristique puis l'améliorer pour qu'elle soit performante, et enfin développer ce jeu en multijoueur, de façon à pouvoir jouer en ligne.

1.2 Outils pour la réalisation du jeu

Ce projet est développé en langage C++, très adapté pour les jeux vidéos. Nous avons décidé de travailler soit sous VM, soit sur un PC avec un environnement Linux. Le projet est développé sous Ubuntu 16.04. Nous avons décidé de coder sous CLION et d'utiliser les bibliothèques SFML pour le rendu.

Nous avons créé un projet GIT en clonant le projet du GIT de M. Barès et nous avons créé nos 4 branches personnelles. De plus, nous avons crée un projet Trello afin de mieux visualiser les tâches restantes.

1.3 Archétype du jeu

L'objectif de notre projet est la réalisation d'un jeu vidéo de type tactical RPG (T-RPG), en tour par tour, basé sur le jeu Donjon de Naheulbeuk.

FIGURE 1.1 – Logo de Donjon de Naheulbeuk

1.4 Règles du jeu

1.4.1 Les personnages et leurs maps

Nous avons créé nos propres personnages et nos propres règles.
Chaque joueur doit choisir une équipe de 3 personnages parmi les suivants : Elfe, Indien, Nain, Bandit, Chevalier, Troll et Pirate.

Le jeu comporte 7 maps représentant les 7 thèmes associés aux personnages :

1. Jungle (elfe)
2. Plaine (indien)
3. Montagne (nain)
4. Forêt (bandit)
5. Château (chevalier)
6. Grotte (troll)
7. Bateau (pirate)

.

.

(a) Elfe (b) Indien (c) Nain

(d) Bandit (e) Chevalier (f) Troll

(g) Pirate

FIGURE 1.2 – Liste des personnages possibles

1.4.2 Les attributs principaux, secondaires et les compétences

A chaque personnage est associé des attributs principaux et secondaires. La répartition des points pour chaque attribut principal se fait par un système d'achat de points :

Attributs principaux
$8 \leq Force \leq 15$
$8 \leq Adresse \leq 15$
$8 \leq Endurance \leq 15$
$8 \leq Courage \leq 15$
$8 \leq Intelligence \leq 15$
$8 \leq Arcane \leq 15$

.

Tous les attributs principaux sont à 8 par défaut et le joueur dispose de 27 points pour améliorer les capacités de son choix, chaque capacité n'excédant pas 15. Si un attribut dépasse 13, il faut alors 2 pts pour passer à 14, puis à 15.

Les points attribués aux attributs secondaires sont calculés automatiquement en fonction des attributs primaires de la manière suivante :

Attributs secondaires	
Points De Vie	3 * Endurance + 2 * Force
Precision	(Adresse + Force + Intelligence + Arcane)/60
Esquive	((Adresse + Intelligence - 16)^2/196) * 0,33
Mouvement	5 + $\lfloor$ 1/12 $\rfloor$ * endurance + $\lfloor$ 1/12 $\rfloor$ * Courage
Initiative	Courage + Intelligence + Arcane

.

Il est aussi possible d'utiliser les compétences des personnages, établies comme suit :

Corps-à-corps	**A distance**	**Magie**
- 2 pt d'actions (ts ls 3 tours) *Le joueur adv ne joue pas*	Pluie de projectiles (3) *Joueur adv essuie 3 attaques*	Téléportation (3)
Pas de mouvement (2)	Rebonds (2) *L'attaque rebondit 2 fois*	Désarmement (2)

.

Chaque compétence vaut 1 point d'action et ne dure qu'un tour. Ces compétences ont un temps de rechargement exprimé en nombre de tours (voir tableau ci-dessus).

Bonus thématique
De plus, si un personnage joue dans la map de même thème (ex : Elfe dans la jungle), alors le personnage se voit attribuer un bonus de 5 points de vie.

Bonus Taxon
De même, chaque personnage a un bonus lié à son origine, selon le tableau ci-dessous :

Bonus de Taxon	
Elfe	+1 en Adresse
Indien	+1 en Arcane
Nain	+1 en Endurance
Bandit	+1 en Intelligence
Troll	+1 en Force
Chevalier	+1 en Courage
Pirate	+1 au choix dans un attribut primaire (car c'est un voleur)

1.4.3 Le tour par tour

A chaque tour, le joueur dispose de 2 points d'action. Pour chaque point, il peut soit se déplacer, soit attaquer, soit utiliser une compétence. Pour chaque tour, il n'est pas possible de réaliser une attaque et une compétence ou deux attaques.

Les attaques

Trois types d'attaques sont disponibles : le corps-à-corps, attaque à distance, ou attaque magique.

Pour chaque arme, une portée minimale et maximale est donnée dans le tableau ci dessous :

Types d'attaques								
Corps-à-corps			**A distance**			**Magique**		
	Portée			**Portée**			**Portée**	
Arme	Min	Max	**Arme**	Min	Max	**Arme**	Min	Max
Epée	1	2	Arc	2	8	Baguette	3	4,5
Hache	1	1,5	Arbalette	2	7	Bâton	3	4
Lance	1	2,5	Fronde	2	6,5	Bracelets	3	5

.

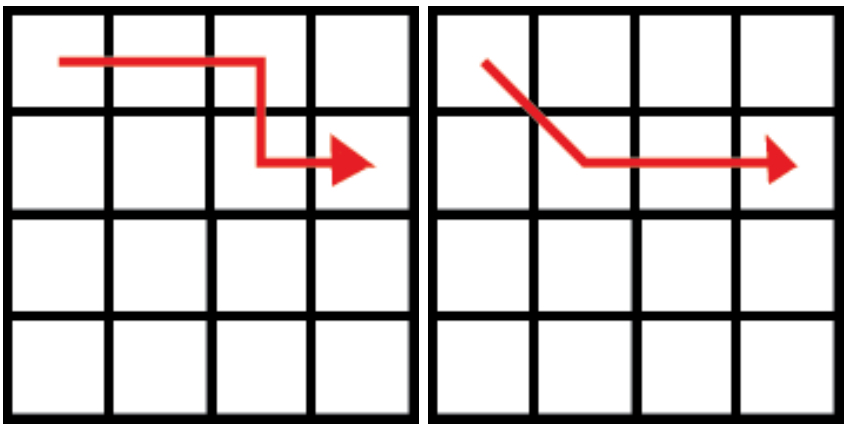
Voici les dégâts causés par ces armes :

Arme	Dégâts
Epée	13
Hache	14
Lance	12
Arc	6
Arbalette	7
Fronde	8
Baguette	10
Bâton	11
Bracelet	9

.

Les déplacements

Chaque personnage peut se déplacer de 5 unités de distance : selon la verticale et/ou horizontale pour une valeur de 1 unité par case, et en diagonale pour une valeur de 1,5 unité de distance par case. Le joueur peut également avoir un bonus de 1 ou 2 unité de distance s'il choisit de maxer l'attribut secondaire "Mouvement".
Exemple :



(a) 4 cases de distance (b) 3,5 cases (1,5+2)

1.5 Textures

FIGURE 1.4 – Texture des personnages

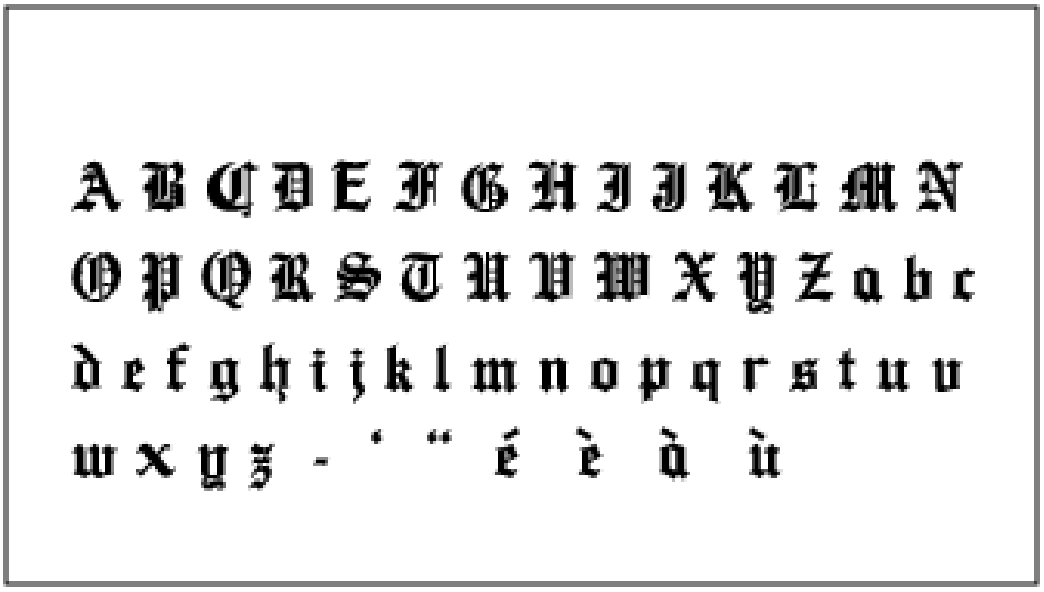


FIGURE 1.5 – Texture de la police

FIGURE 1.6 – Texture du fond

Chapitre 2

Description et conception des états

2.1 Etat du jeu

Un état du jeu est formée par deux équipes de personnage (entre 3 et 6 personnages par équipe) et une map. Tous les éléments (map et personnages) sont représentés par des carrés composants la map. Ces carrés possèdent :
— Une coordonnée (x,y)
— Un identifiant

Les map sont au nombre de 7, rectangulaires et de tailles identiques.

Le fond

Il y a toutes sortes de texture qui s'adaptent aux décors voulu ; texture de forêt dense pour la jungle, de l'herbe pour les plaines, des rochers pour les montagnes, des arbres pour la forêt, des pierres pour le château fort, de la terre pour la grotte, et des planche de bois pour le sol du bateau. La map constitue le premier layer du rendu.

Les décors

Par ailleurs, des décors seront apposé sur cette texture de fond (comme des tonneaux, des trons d'arbres ou des rochers imposants. Ces décors seront infranchissable par les personnages et bloqueront la portée de leur arme. Les décors contituent le deuxième layer du rendu.

Les personnages

Les différents personnages n'intéragissent pas avec la texture de fond. Ils constituent le 3ème layer du rendu.

2.2 Diagramme de classes

Le diagramme des classes est représenté ci-dessous. Voici les classes qui le compose :

(Dans chaque classe, on retrouve un constructeur et un destructeur ainsi que les setters et les getters, car nos attributs sont privés)

Classe Player

Chaque joueur à un *name*, un *Id* et une *ListCharacters* qui contient entre 3 et 6 personnages.

Classe Character
Chaque personnage possède : un *TypeID*, dont les valeur sont données grâce à une énumération. On utilise les énumération pour optimiser le code. Il en est de même pour les attributs *Status stats* et *direction*.
Un personnage possède également un *Name*, un *Movement* pour donner son nombre de case de déplacement maximal, un *Health* pour ses points de vie, des attributs *Dogde* et *Precision* pour la capacité d'esquive et de présicion lors d'un attaque, un *effect* qui dépend de la classe Effect, et pour finir un *charWeap* qui permet à chaque personnage de porter une arme.

Classe Weapon
La classe character dépend de la classe Weapon. Weapon possède les attributs *Owner*, un *typeWeapon* qui utilise l'énumération WeaponID, qui elle-même est en charge de déterminer le type d'arme utilisé (Sword, axe, spear ...).

Pour la classe Weapon, on aurait pu faire une interface et faire hériter chaque arme de la classe mère, et utiliser le polymorphisme, mais l'énumération était plus rapide et tout aussi performante.

Classe Effect
Lorsqu'un personnage subit une attaque, celui-ci passe dans un état qui peut être étourdit, brulé... Cet état est représenté dans la classe Efect.

state
State
-listPlayers: std::vector<std::unique_ptr<Player>>
-map: std::vector<std::vector<std::unique_ptr<MapCell>>>
-endGame: bool
-cursor: Cursor
-round: int
-curActionID: CurActionID
-curPlayerID: int
-nbOfPlayers: int
-gameWinner: int
-mode: std::string
+State()
+~State()
+getListPlayers(): std::vector<std::unique_ptr<Player>>&
+getListCharacters(playerID:int): std::vector<std::unique_ptr<Character>>&
+getMap(): std::vector<std::vector<std::unique_ptr<MapCell>>>&
+getEndGame(): bool
+getCursor(): Cursor&
+getRound(): int
+getCurAction(): CurActionID
+getCurPlayerID(): int
+getNbOfPlayers(): int
+setRound(newRound:int): void
+setEndGame(res:bool): void
+setCurPlayerID(newPlayer:int): void
+setCurAction(newAction:CurActionID): void
+deletePlayer(player:Player&): void
+ifSelected(): bool
+initPlayers(): void
+initCharacters(): void
+initMapCell(): void
+setGameWinner(winnerID:int): void
+getGameWinner(): int
+setMode(newMode:std::string): void
+getMode(): std::string
Observable
-observers: std::vector<Observer*>
+Observable()
+registerObserver(observer:Observer*): void
+notifyObservers(stateEvent:const StateEvent&,
state:State&): void const
+unregisterObserver(obs:Observer*): void
+~Observable()
<<interface>>
Observer
+stateChanged(stateEvent:const state::StateEvent&,
aState:State&): void
<<enumeration>>
CurActionID
+IDLE = 1
+MOVING = 2
+ATTACKING = 3
Position
-x: int
-y: int
+Position()
+~Position()
+Position(x:int,y:int)
+setX(): int
+getY(): int
+setX(x:int): void
+setY(y:int): void
+equals(position:Position&): bool
+distance(position:Position&): int
+getNearPositions(): std::vector<Position>
Element
#name: std::string
#position: Position
#tileCode: int
+Element()
+isMapCell(): bool
+setName(): std::string
+setPosition(): Position&
+~Element()
+getTileCode(): int
+setTileCode(a tileCode:int): void
+equals(other:Element&): bool
StateEvent
+stateEventID: StateEventID
+text: std::string
+infoText: std::string
+stateText: std::string
+StateEvent(eventID:StateEventID)
+setStateEvent(newID:StateEventID): void
Player
-Name: std::string
-Id: int
-listCharacters: std::vector<std::unique_ptr<Character>>
+~Player()
+Player(id:int)
+Player(id:int,name:std::string)
+setName(n:std::string): void
+setName(): std::string
+setID(): int
+setListCharacters(): std::vector<std::unique_ptr<Character>>&
+setID(id:int): void
+deleteCharacter(character:Character&): void
+addCharacter(ptr_character:std::unique_ptr<Character>): void
Weapon
#Owner: std::string
#TypeWeapon: WeaponID
#Damages: int
#MaxRange: float
#MinRange: float
+Weapon(id:WeaponID)
+~Weapon()
+setOwnerName(n:std::string): void
+setOwner(): std::string
+setDamages(dmg:int): void
+setDamages(): int
+setMaxRange(orM:float): void
+setMaxRange(): float
+setMinRange(orm:float): void
+setMinRange(): float
+setType(): WeaponID
Character
#TypeID: CharacterTypeID
#Name: std::string
#Status: CharacterStatusID
#Movement: int
#Health: int
#Dodge: float
#Precision: float
#effect: Effect
#CharWeapon: Weapon*
#stats: Stats
#PlayerID: int
#direction: Direction
#MovementLeft: int
#index: int
+Character(id:CharacterTypeID,name:std::string,
x:int,y:int,a tileCode:int=0)
+~Character()
+getTypeID(): CharacterTypeID
+setPlayerID(): int
+setName(): std::string
+setStatus(): CharacterStatusID
+setMovement(): int
+setMovementBonus(Courage:int,Stamina:int): void
+setHealth(): int
+setHealth(Stamina: int,Strength: int): void
+setDodge(): float
+setDodge(Agility:int,Intelligence:int): void
+setPrecision(): float
+setPrecision(Agility: int,Intelligence:int,
Strength:int,Arcane:int): void
+setDirection(): Direction
+setDirection(dir:Direction): void
+setEffect(): Effect&
+setEffect(Immobilised:bool,Stunned:bool,
Disarmed:bool): void
+setCharWeapon(w:Weapon*): void
+setCharWeapon(): Weapon*
+setStats(Agility:int,Arcane:int,Courage:int,
Intelligence:int,Stamina:int,Strength:int): void
+setStats(): Stats&
+setMovementLeft(): int
+setMovementLeft(movement:int): void
+receiveDamage(damage:float): void
+allowedMove(state:State&): std::vector<Position>
+allowedAttackPos(state:State&): std::vector<Position>
+isMapCell(): bool
+allowedAttackTarget(state:State&): std::vector<int>
+setHealWith(newHealth:int): void
Cursor
-visible: bool
-lastPosition: Position
+Cursor(x:int=0,y:int=0,a tileCode:int=0)
+isMapCell(): bool
+moveIdestination:Position&): void
+setVisible(): bool
+setVisible(visibility:bool): void
+setLastPosition(): Position&
MapCell
+isMapCell(): bool
+isOccupied(state:State&): int
+isSpace(): bool
Effect
#Immobilised: bool
#Stunned: bool
#Disarmed: bool
+Effect()
+~Effect()
+setStunned(s:bool): void
+setImmobilised(i:bool): void
+setDisarmed(d:bool): void
+setStunned(): bool
+setImmobilised(): bool
+setDisarmed(): bool
Stats
#Agility: int
#Arcane: int
#Courage: int
#Intelligence: int
#Stamina: int
#Strength: int
+Stats()
+~Stats()
+Stats(Agility:int,Arcane:int,Courage:int,
Intelligence:int,Stamina:int,Strength:int)
+setAgility(): int
+setArcane(): int
+setCourage(): int
+setIntelligence(): int
+setStamina(): int
+setStrength(): int
+setAgility(Agility:int): void
+setArcane(Arcane:int): void
+setCourage(Courage:int): void
+setIntelligence(Intelligence:int): void
+setStamina(Stamina:int): void
+setStrength(Strength:int): void
ObstacleMapTiles
#TypeID: ObstacleMapTilesID
+ObstacleMapTiles(id:ObstacleMapTilesID,
newX:int,newY:int,a tileCode:int=0)
+~ObstacleMapTiles()
+isSpace(): bool
+setObstacleMapTilesID(): ObstacleMapTilesID
+setObstacleTilesID(id:ObstacleMapTilesID): void
SpaceMapTiles
#TypeID: SpaceMapTilesID
+SpaceMapTiles(id:SpaceMapTilesID,newX:int,
newY:int,a tileCode:int=0)
+~SpaceMapTiles()
+setSpaceMapTilesID(): SpaceMapTilesID
+isSpace(): bool
+setSpaceMapTilesID(id:SpaceMapTilesID): void
<<enumeration>>
WeaponID
+SWORD = 1
+AXE = 2
+SPEAR = 3
+BOW = 4
+CROSSBOW = 5
+SLING = 6
+WAND = 7
+STICK = 8
+STAFF = 9
<<enumeration>>
StateEventID
+ALLCHANGED = 0
+ROUNDCHANGED = 1
+CHARCHANGED = 2
+CURSORCHANGED = 3
+ENDGAME = 4
<<enumeration>>
CharacterTypeID
+CROOK = 1
+KNIGHT = 2
+ELF = 3
+NATIVE = 4
+DWARF = 5
+PIRATE = 6
+TROLL = 7
<<enumeration>>
ObstacleMapTilesID
#Hole = 1
#Wall = 2
#Rock = 3
#Water = 4
#Fire = 5
<<enumeration>>
SpaceMapTilesID
#Grass = 1
#Rocks = 2
#Sand = 3
#Plants = 4
<<enumeration>>
Direction
+UP = 1
+RIGHT = 2
+DOWN = 3
+LEFT = 4
<<enumeration>>
CharacterStatusID
+SELECTED = 1
+AVAILABLE = 2
+WAITING = 3
+DEATH = 4
+TARGET = 5

Chapitre 3

Rendu : Stratégie et Conception

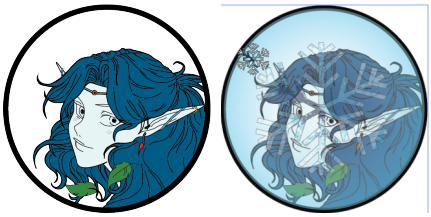
Pour le rendu de l'application : nous avons utilisé Adobe Illustrator pour réaliser les personnages, les états des personnages (étourdissement, immobilité, désarmement), ainsi que le logo Zorglub et l'initialisation du jeu.
Pour les autres décors et les bouttons nous avons utilisé la bibliothèque SFML.

Pour le rendu d'un état, nous avons utilisé 3 layers : un plan pour le sol, selon les différentes map ; un plan pour les différents obstacles que les personnages peuvent rencontrer ; et un plan pour les personnages.

La classe Wallpaper permet de choisir le fond d'écran que l'on va appliquer sur la fenêtre de jeu. Deux autres layers vont se supperposer à Wallpaper (voir ressources).

La classe TileSet permet de récuperer des tuiles des ces deux layers. Cette classe est

Pour chaque état, on met à jours la position du personnage et son état (effet des compétences). On met également à jours les points de vie des personnages et les points d'actions.



(a) Elfe sans effets

(b) Elfe immobilisée

(c) Elfe désarmée

(d) Elfe étourdie

FIGURE 3.1 – Liste des effets possibles sur les personnages

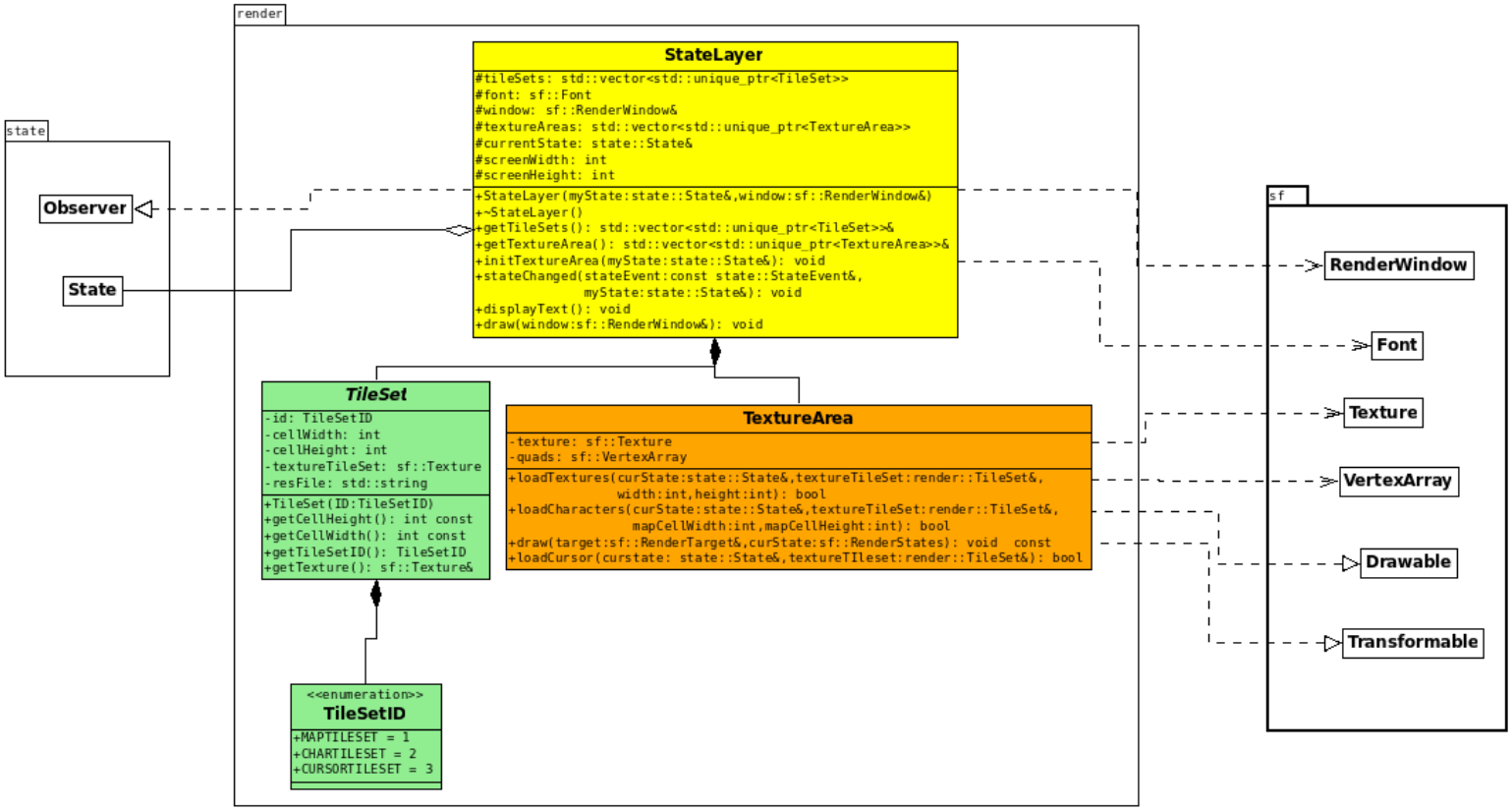


FIGURE 3.2 – Diagramme de rendu du jeu

Chapitre 4

Moteur de Jeu

4.1 Action de l'utilisateur

Le joueur dispose d'un menu, sur lequel il peut choisir (avec un clic de souris sur un boutton) d'attaquer, d'utiliser une compétence, ou de se déplacer. Il peut :
- Attaquer et se déplacer
- Se déplacer et attaquer
- Utiliser sa compétence et se déplacer
- Se déplacer et utiliser sa compétence
- Se déplacer deux fois

S'il attaque ou s'il utilise sa compétence, le joueur clique sur le boutton associé et il doit choisir la case sur laquelle il veut l'utiliser avec un clic de souris.

De même, s'il choisit de se déplacer, il clique sur le boutton associé et il doit choisir sur quelle case il veut aller, dans la limite des cases dont il a le droit.

4.2 Actions internes

Chaque point d'action est immédiatement exécuté. Lors d'un déplacement, la position du joueur est immédiatement actualisée ; lors d'une attaque, les Points de vie de l'adversaire sont diminués, de même pour les compétences. Un compteur est incrémenté à chaque point de vie. C'est utile pour les compétences ; certaines ne peuvent être utilisées que tout les 2 ou 3 tours.

Enfin on vérifie que chaque joueur possède au moins encore un personnage capable de se battre, c'est à dire avec des Points de vie non nul. Ceci est vérifié avec la fonction Check_Win.

4.3 Conception logiciel

Classe Command
Cette classe est une classe abstraite qui décrit n'importe quelle commande.

CommandID
La classe commandID énumère les différentes commandes.

Move_Command
Cette classe commande le déplacement.

Attack_Command
La classe Attack_Command est la classe de commande d'attaque.

Classe Engine
Cette classe gère les états successifs du jeu et les commandes à executer.

Classe engineObservable
Cette classe est la classe qui rend compte des diverses commandes effectuées.

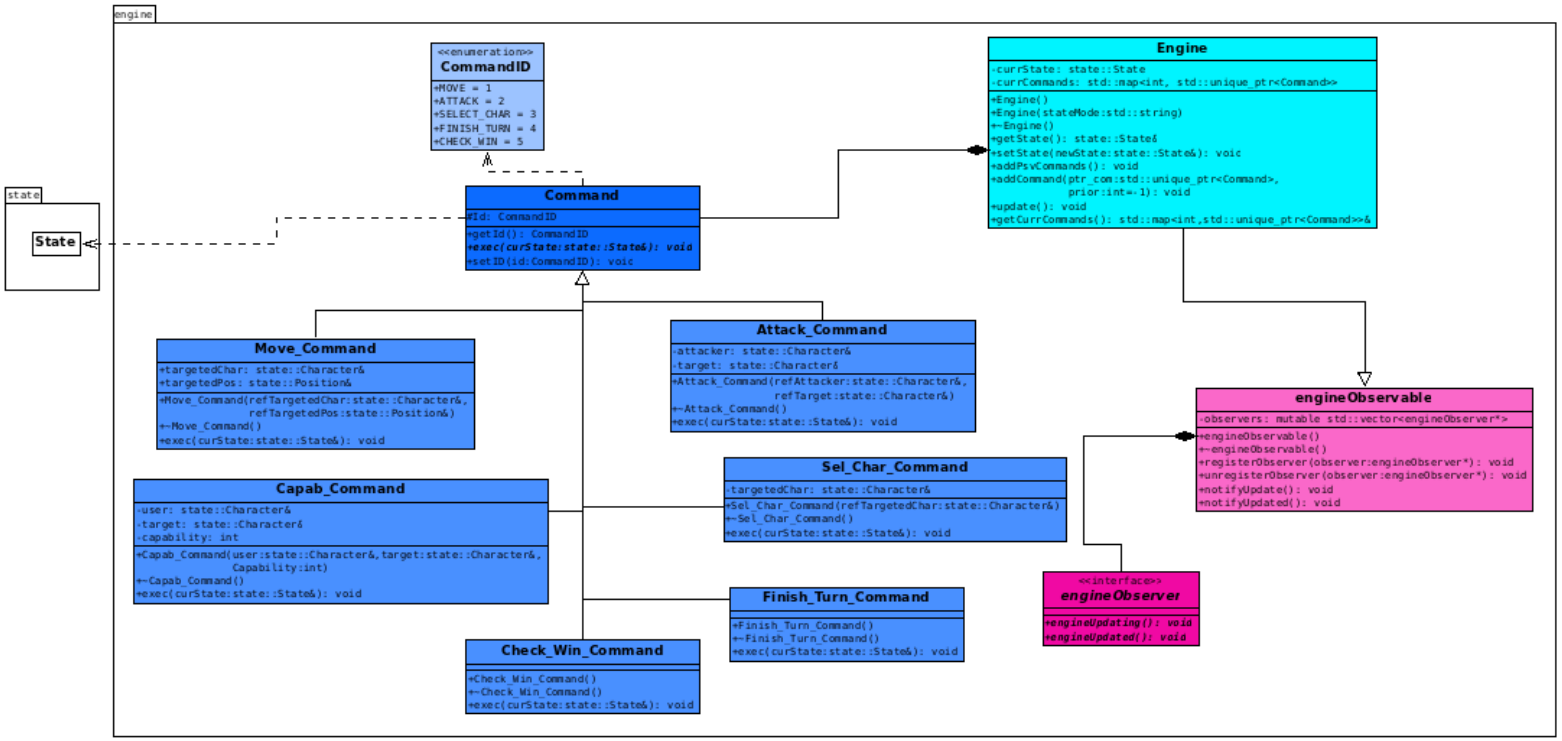


FIGURE 4.1 – Diagramme engine

4.4 Intelligence artificielle

4.4.1 Intelligence artificielle random

4.4.2 Intelligence artificielle heuristique

4.4.3 Intelligence artificielle avancée

Bibliographie