Projet Final Mystic Quest

Abdallah OUSAID – Sandoche BARRES

Figure 1 : Illustration du jeu Final Fantasy (map)

Figure 2 : Illustration du jeu Final Fantasy (Combat)

Objectif

1. Présentation générale

L'objectif de ce projet est de réaliser un clone du célèbre jeu de rôle Final Fantasy avec les commandes et les règles de bases. Un exemple est présenté sur les figures 1 et 2. Le but du jeu est de récupérer 4 cristaux que protègent les boss dans chaque donjon.

2. Règles du jeu

Final Fantasy est un RPG (Rôle Playing Game) où l'on contrôle un personnage qui navigue sur une map en 2D. On peut naviguer dans trois types d'environnements qui sont : la map du monde, les villes et les donjons. Dans la map du monde et les donjons on rencontre des ennemis qui sont visibles sur la carte ce qui nous donne une liberté d'engager ou non le combat. Dans les villes on récupère principalement des objets, des équipements et de la vie. Dès qu'on rentre dans une phase de combat, on change de plan visuel. L'équipe du joueur se déploie en une équipe de deux ou trois personnages contrôlés par ce dernier. Ce jeu se déroule en deux phases :

Phase de combat :

Chaque personnage va effectuer une action à tours de rôles (attaquer, magie, se soigner, etc). Dès que tous les personnages ont fini de prendre leurs décisions et qu'elles seront validées par le joueur, elles vont s'effectuer à tour de rôle ainsi que celles de l'ennemi pris par l'intelligence artificielle. A chaque combat gagné, l'équipe gagne en expériences et son butin. Enfin on sort de la phase de combats après dès que l'on a détruit tous les monstres. Les personnages et les ennemies ont une vie que l'on nomme HP (Health Power) caractérisé par un numéro entre 0 et 9999 selon le niveau des personnages et ennemies. Quand le personnage a 0 de HP la partie se termine.
En plus de la vie, chaque personnage et ennemies possèdent un niveau d'expérience et des statistiques qui leurs sont propres. Après chaque combat gagné, l'expérience permet d'augmenter ce niveau et ces statistiques alors que ceux des ennemies sont déjà fixés.

Phase de navigation :

Le joueur se balade dans la carte du monde et choisit soit de rentrer dans une ville, soit de rentrer dans un donjon ou soit de rentrer dans une serre de combat pour augmenter ses niveaux et gagner de l'expérience et des objets.

Univers du jeu : Le jeu se déroule dans un univers médiéval-fantastique composé de donjon et de village.

Héro et compagnon : On incarne un chevalier qui manie plusieurs types d'arme et qui a les statistiques initiales les plus élevées. Il rencontrer au cours de sa quête des compagnons qui ne peuvent manipuler qu'un seul type d'arme. Ils lui prêteront main forte dans sa quête des cristaux.

Ennemis et Boss : Les ennemis sont un ensemble de monstre mystique qui ont les mêmes attributs que les personnages du jeu sauf qu'ils sont commandés par l'intelligence artificielle. Le boss garde chaque donjon et détient un cristal.

Description et conception des états

2.1 Description des états :

Notre état du jeu est constitué de deux univers : nous avons le monde et le plan de combat. Le monde se compose d'un ensemble d'élément fixe (des donjons, les villes et les serres de combats).

Les donjons : sont les éléments principaux du jeu car ils contiennent les cristaux nécessaire pour le finir. Le donjon est constitué d'éléments fixes (coffres d'objets, ennemis/boss et le labyrinthe) et d'éléments mobiles (joueur).

Villes : sont les éléments du jeu ou l'on peut régénérer de la vie, acheter des objets et des équipements. Ils constituent comme les donjons des petits mondes en elles-mêmes.

Serres de combats : c'est un passage direct de la map du monde à la phase de combats sans que cela influence l'évolution du jeu. Ce lieu permet de faire gagner de l'expérience et des objets grâce aux combats menés avec le libre arbitre du joueur

Chacun de ces trois éléments possèdent :

- Coordonnées (x ; y) dans la map.
- Un identifiant de type d'éléments : ce nombre indique la nature de l'élément.

2.2 Eléments des états :

Chaque lieu du jeu (monde, donjons, villes et serres de combats) sont des mini monde en eux-mêmes. Ils se composent d'éléments fixes qui sont :

Les ennemis : se trouvent seulement dans les donjons et serres de combats sous la forme d'un GIF animé.
Les coffres d'objets : se trouvent un peu partout dans les villes et les donjons. Ces coffres peuvent contenir des équipements, fioles de vie, fioles de magies et des objets.
Les obstacles : il s'agit des éléments infranchissables par le joueur. Il peut s'agir par exemple des arbres, montagnes, murs.
Les murs : ce sont les éléments qui délimitent les espaces du jeu (donjons, villes et carte de monde) et qui permettent de contenir tous les éléments du jeu dans un espace restreint.
Les espaces de départs: qui définissent les positions initiaux pour le joueur dans différents lieux du jeu.

Éléments mobiles :

L'élément mobile du jeu, qui correspond uniquement au personnage, possède une direction (aucune, gauche, droite, haut ou bas), une position et une clef de localisation qui permet de définir la localisation géographique du personnage.

L'élément mobile « personnage » est dirigé par le joueur et celui-ci commande la propriété de direction de ce personnage. Il possède deux status :

- Mode navigateur : où le personnage se déplace dans la carte ou les autres lieux de navigation.
- Mode combat : où le personnage peut déployer ses armes et compétences.
- Mort : lorsque le personnage a 0 de point de vie.

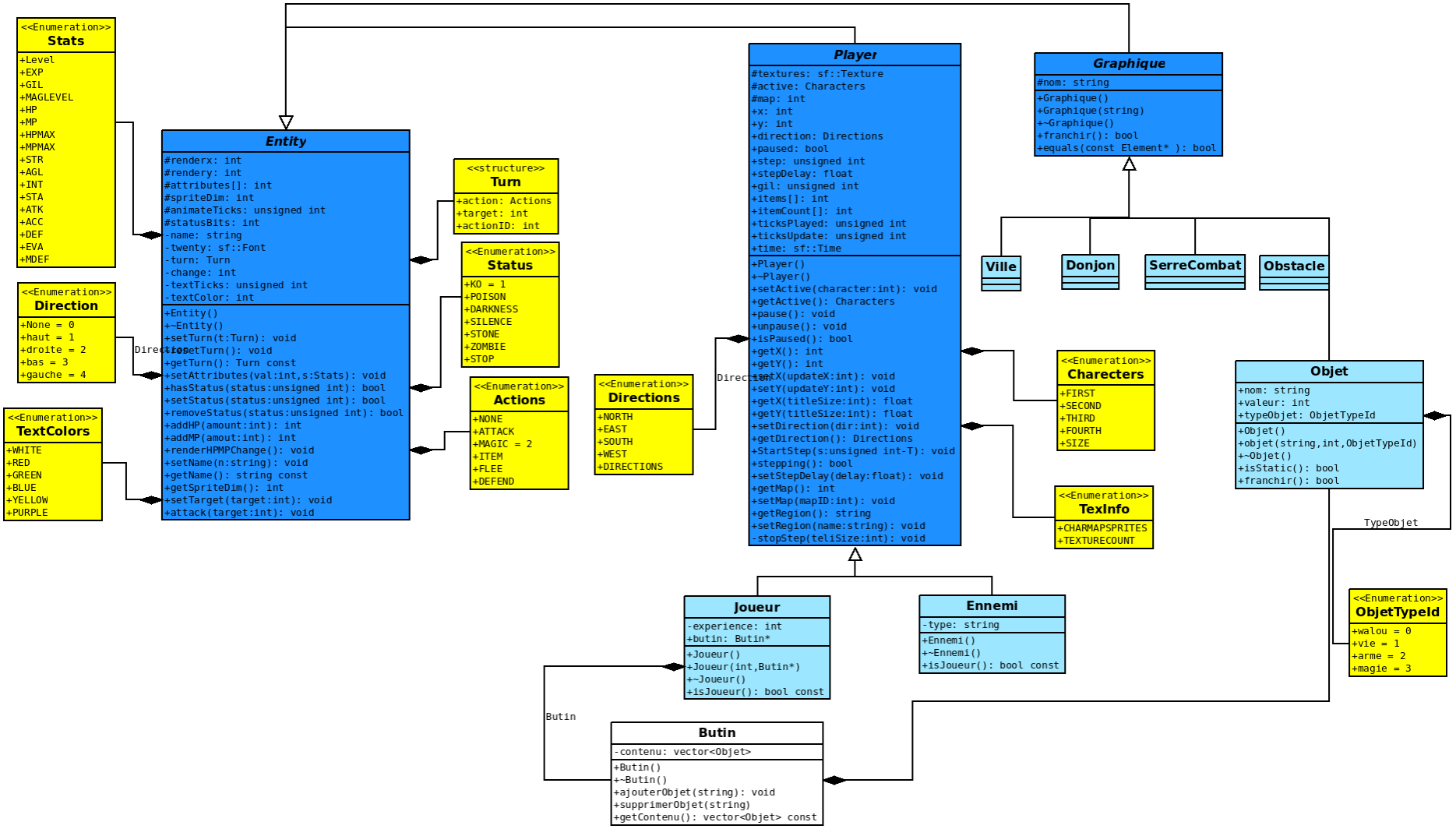
État général :

Le personnage possède un niveau d'expérience qui évolue selon le nombre de combat gagné et récupère des cristaux à chaque combat gagné contre le boss qui se trouve à la fin d'un donjon. Dès que le joueur a récupéré quatre cristaux le jeu se termine.

1. Conception logiciel :

Diagramme UML :

Le diagramme UML suivant résume l'organisation des classes de notre jeu. Cela va nous permettre de modéliser les différentes classes et donc nous faciliter la conception et détecter facilement les éventuels bugs. Cette réalisation risque de changer lors de la détection de bugs en phase de réalisation.

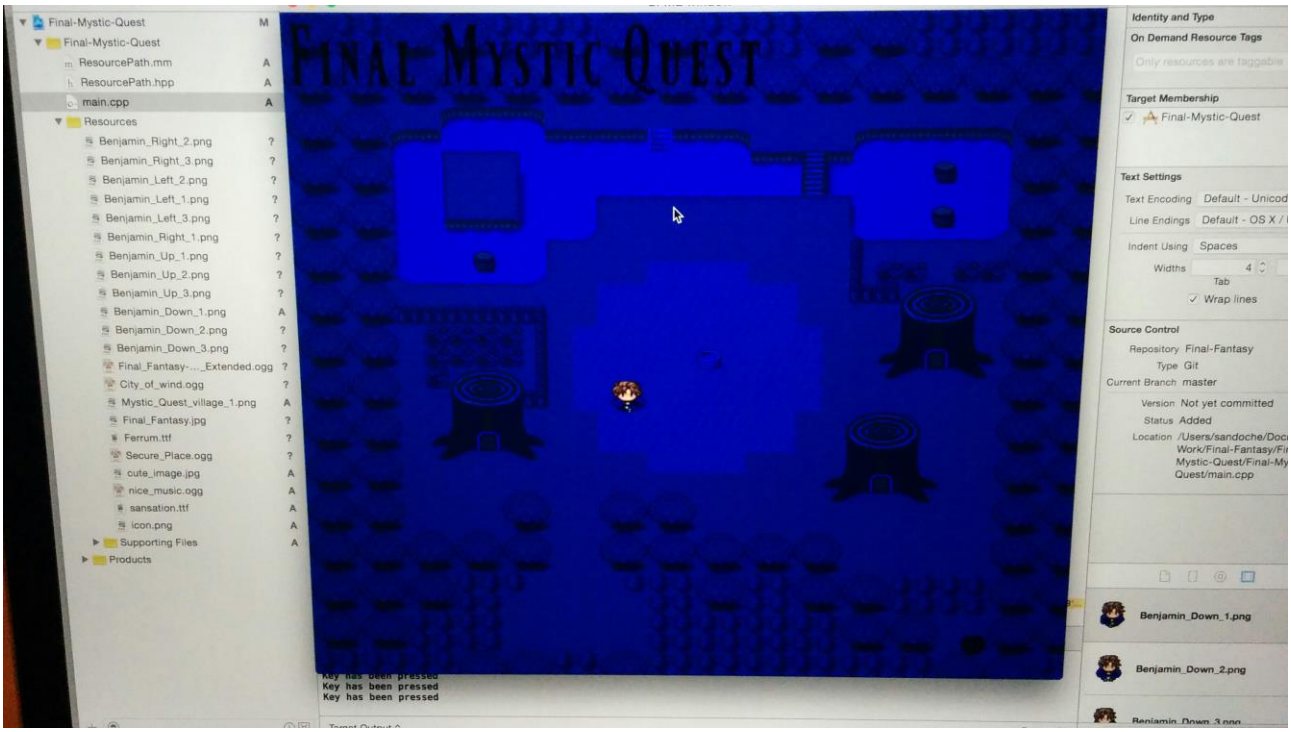

<<Enumeration>>
Stats
+Level
+EXP
+GIL
+MAGLEVEL
+HP
+MP
+HPMAX
+MPMAX
+STR
+AGL
+INT
+STA
+ATK
+ACC
+DEF
+EVA
+MDEF
<<Enumeration>>
Direction
+None = 0
+haut = 1
+droite = 2
+bas = 3
+gauche = 4
<<Enumeration>>
TextColors
+WHITE
+RED
+GREEN
+BLUE
+YELLOW
+PURPLE
Entity
#renderx: int
#rendery: int
#attributes[]: int
#spriteDim: int
#animateTicks: unsigned int
#statusBits: int
-name: string
-twenty: sf::Font
-turn: Turn
-change: int
-textTicks: unsigned int
-textColor: int
+Entity()
+~Entity()
+setTurn(t:Turn): void
+resetTurn(): void
+getTurn(): Turn const
+setAttributes(val:int,s:Stats): void
+hasStatus(status:unsigned int): bool
+setStatus(status:unsigned int): bool
+removeStatus(status:unsigned int): bool
+addHP(amount:int): int
+addMP(amout:int): int
+renderHPMPChange(): void
+setName(n:string): void
+getName(): string const
+getSpriteDim(): int
+setTarget(target:int): void
+attack(target:int): void
<<structure>>
Turn
+action: Actions
+target: int
+actionID: int
<<Enumeration>>
Status
+KO = 1
+POISON
+DARKNESS
+SILENCE
+STONE
+ZOMBIE
+STOP
<<Enumeration>>
Actions
+NONE
+ATTACK
+MAGIC = 2
+ITEM
+FLEE
+DEFEND
<<Enumeration>>
Directions
+NORTH
+EAST
+SOUTH
+WEST
+DIRECTIONS
Player
#textures: sf::Texture
#active: Characters
#map: int
+x: int
+y: int
+direction: Directions
+paused: bool
+step: unsigned int
+stepDelay: float
+gil: unsigned int
+items[]: int
+itemCount[]: int
+ticksPlayed: unsigned int
+ticksUpdate: unsigned int
+time: sf::Time
+Player()
+~Player()
+setActive(character:int): void
+getActive(): Characters
+pause(): void
+unpause(): void
+isPaused(): bool
+getX(): int
+getY(): int
+setX(updateX:int): void
+setY(updateY:int): void
+getX(titleSize:int): float
+getY(titleSize:int): float
+setDirection(dir:int): void
+getDirection(): Directions
+StartStep(s:unsigned int-T): void
+stepping(): bool
+setStepDelay(delay:float): void
+getMap(): int
+setMap(mapID:int): void
+getRegion(): string
+setRegion(name:string): void
-stopStep(teliSize:int): void
Graphique
#nom: string
+Graphique()
+Graphique(string)
+~Graphique()
+franchir(): bool
+equals(const Element*): bool
Ville
Donjon
SerreCombat
Obstacle
<<Enumeration>>
Charecters
+FIRST
+SECOND
+THIRD
+FOURTH
+SIZE
<<Enumeration>>
TexInfo
+CHARMAPSPRITES
+TEXTURECOUNT
Objet
+nom: string
+valeur: int
+typeObjet: ObjetTypeId
+Objet()
+objet(string,int,ObjetTypeId)
+~Objet()
+isStatic(): bool
+franchir(): bool
<<Enumeration>>
ObjetTypeId
+walou = 0
+vie = 1
+arme = 2
+magie = 3
Joueur
-experience: int
+butin: Butin*
+Joueur()
+Joueur(int,Butin*)
+~Joueur()
+isJoueur(): bool const
Ennemi
-type: string
+Ennemi()
+~Ennemi()
+isJoueur(): bool const
Butin
-contenu: vector<Objet>
+Butin()
+~Butin()
+ajouterObjet(string): void
+supprimerObjet(string)
+getContenu(): vector<Objet> const
Direction
Direction
TypeObjet
Butin

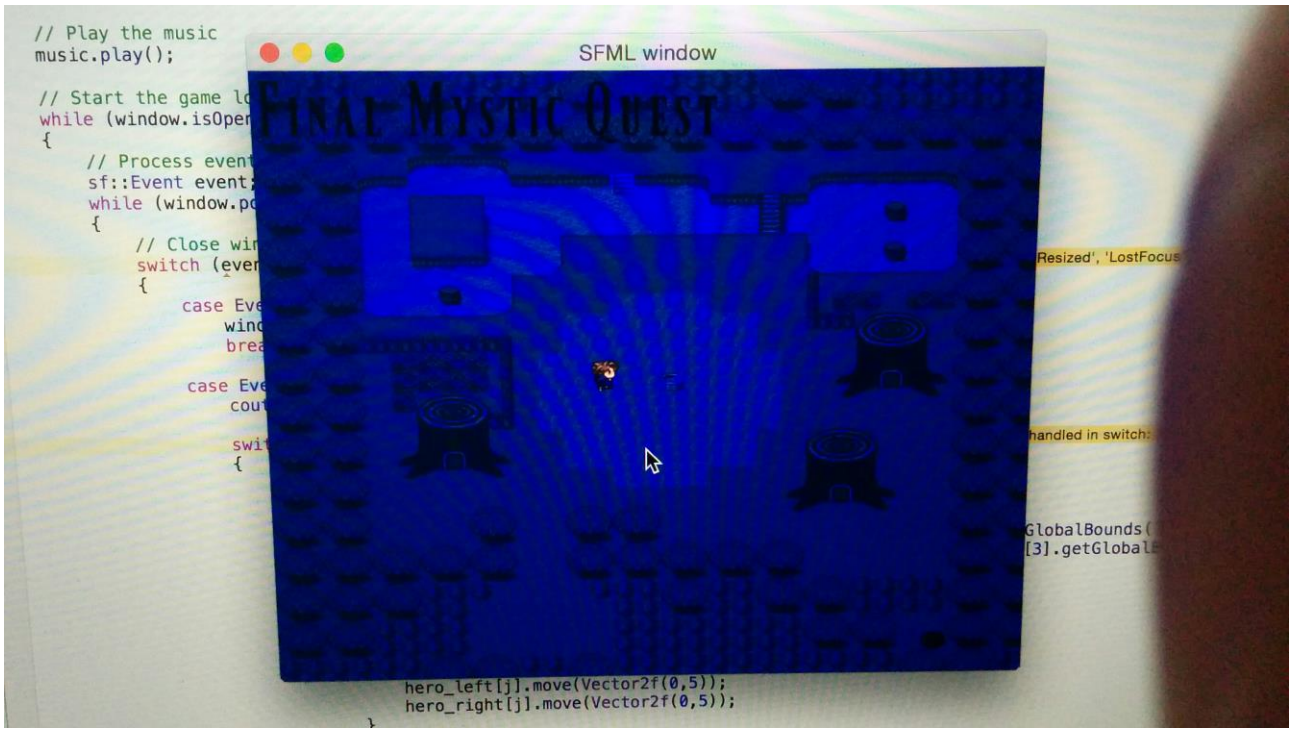
4. Rendu : Stratégie et Conception :

4.1 Stratégie de rendu d'un état

Pour le rendu d'un état, nous n'avons pas choisi de stratégie particulière. Comme dis plus haut, le jeu se décompose en deux phases : une phase exploration et une phase de combat. Pour le rendu d'un état, nous avons choisi d'afficher pour le moment le jeu dans sa phase d'exploration. Pour cela, nous avons affiché un village dans sa globalité avec le sprite du héros qui se déplace dans le village. Nous avons donc mis en premier plan le sprite du village et celui du héros. Aucun élément de collision n'a été mis en évidence. Par contre les différents états de déplacements du héros ont été mis en place. Pour cela nous avons chargé chaque états du héros dans une texture différente et ensuite afficher les sprites. Les touches directionnelles du clavier servent à afficher ces différents états. Par exemple en appuyant la touche directionnelle gauche, le héros regarde vers la gauche. Pour permettre une détection en temps réelle des différents changements d'états du personnage, une structure avec des switch est utilisée. La méthode utilisée pour afficher un tel changement d'état n'est pas du tout optimisé et elle est même assez laborieuse car une fois qu'un autre event intervient, le sprite disparaît. Il faut donc trouver une autre méthode beaucoup plus efficace pour permettre un affichage en continu. Mais le plus dure à été fait car c'est essentiellement le sprite du personnage que l'on va manipuler dans la suite.

Voici le sprite du village avec le héros qui se déplace:





5. Règles de changement d'états et moteur de jeu :

Pour le moteur de jeu, son rôle est de gérer en toute abstraction des autres classes. En effet, il permet de réunir toutes les règles et les fonctionnalités du jeu indépendamment du rendu qui peut changer de technologie (de SFML à OpenGL par exemple) ou des commandes qui peuvent venir d'une intelligence artificielle ou d'un client Android ou autre. Pour cela, on a réalisé lors du dernier jalon les classes qui permettent d'avoir un rendu satisfaisant. Donc la réalisation de ce jalon est d'autant plus simple car le moteur de rendu est déjà mis en place donc il faut juste créer les classes qui vont représenter le moteur de jeu. En effet, on a amélioré le rendu en utilisant un tableau de vertex. Cette solution permet de diminuer l'utilisation du CPU et GPU car l'utilisation de plusieurs textures sur un même rendu utilise beaucoup de capacité CPU pour pas grand-chose. En plus de cela, cette solution permet d'améliorer le rendu car les events de la souris influence les events du clavier chose non souhaité pour avoir un meilleur affichage et sans bugs. On arrive donc à différencier les deux events et avoir pour le cout un meilleur rendu.
L'optimisation du moteur de jeu est encore sujet d'amélioration car l'abstraction des différents éléments du moteur ne sont pas encore au point. Car cela va nous permettre d'améliorer les fonctionnalités du moteur sans toucher à toutes les structures du jeu. L'abstraction va nous permettre donc de faire la maintenance et l'amélioration avec une grande facilité mais aussi efficace.