Léonard Pannetier MAIN4
Samson Petros Russom

Rapport Projet C++
Coupe du monde

I Description de l'application développée

Notre projet C++ de ce semestre s'appelle Panini GameCard, le but du projet est de coder en C++ un jeu de carte stratégique de duel à deux joueurs dans l'esprit des cartes de Football panini.

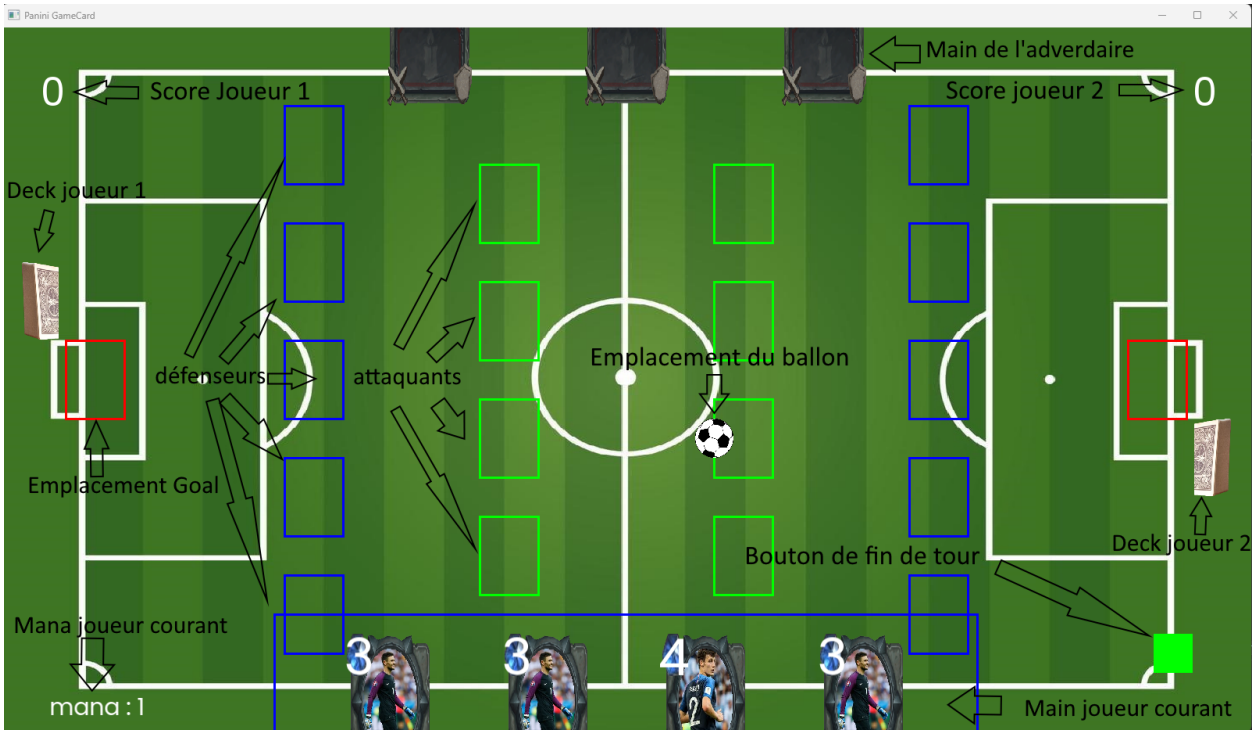
Le jeu est composé :
- D'un plateau de jeu possédant 20 emplacements pour les cartes, commun aux deux joueurs et découpé en 2 parties, le côté gauche est le camp du joueur 1, le droit celui du joueur 2.
Les emplacements rouges sont ceux des goals, les bleus ceux des défenseurs et les verts ceux des attaquants.

Chaque joueur possède :
- Un deck (paquet de cartes)
- Une main (composé de 0 à 5 cartes)
- Des points mana lui servant à jouer des cartes depuis sa main.

Une carte possède plusieurs caractéristiques :
- Un rôle (Goal, Attaquant ou Défenseur)
- Un coût en mana (écrit en haut à gauche de la carte)
- Des points d'attaque (écrit en bas à gauche de la carte)
- Des points de défense (écrit en bas à droite de la carte)
- Une ou plusieurs capacités expliquées par un texte.

But du jeu :
- Comme pour le football, le but est de marquer plus de but que son adversaire, pour cela il faut faire rentrer la balle dans le but adverse.

Début :
- Les deux decks des joueurs sont mélangés puis chaque joueur pioche 3 cartes, le plateau de jeu est vide, seule la balle est placée au centre aléatoirement du côté de l'un des deux joueurs. Les deux joueurs commencent le jeu avec un point de mana.



Phase de jeu :
Le jeu fonctionne au tour par tour, chaque joueur joue un tour sur deux et un joueur ne doit pas regarder l'écran lorsqu'il ne joue pas.

Pendant son tour, le joueur peut faire plusieurs chose :
- Poser une carte de sa main sur le terrain, pour que ce soit possible il faut que :
    - Le joueur possède assez de mana pour jouer sa carte.
    - La carte doit être placée de son côté du terrain.
    - La carte doit être placée dans un emplacement correspondant à son rôle.

- Déplacer une carte du plateau sur une case adjacente.

- Effectuer un duel contre une carte adverse adjacente.

Les duel se passe de cette manière :
- Si l'attaque de la carte attaquante et supérieur à la carte qui défend alors la carte qui défend est détruite, la carte attaquante prend sa place sur le plateau, sinon la carte attaquante est détruite.

Les cartes peuvent effectuer un déplacement ou un duel une fois par tour et ne peuvent pas être utilisées le tour où elles sont posées.

Le ballon est contrôlé par la carte sur le même emplacement, ainsi si une carte en possession du ballon se déplace, alors le ballon bougera se déplacera également.

Si un de vos joueur entre dans le but adverse alors il y a but, vous gagnez 1 points de score et une nouvelle partie se lance.

De plus le deck de chaque joueur se vide, si celui-ci est vide vous ne piocherez plus de carte. De même chaque joueur n'est pas autorisé à avoir plus de 5 cartes en main, si vous avez 5 cartes au début de votre tour, la carte piochée sera détruite au lieu d'aller dans votre main.

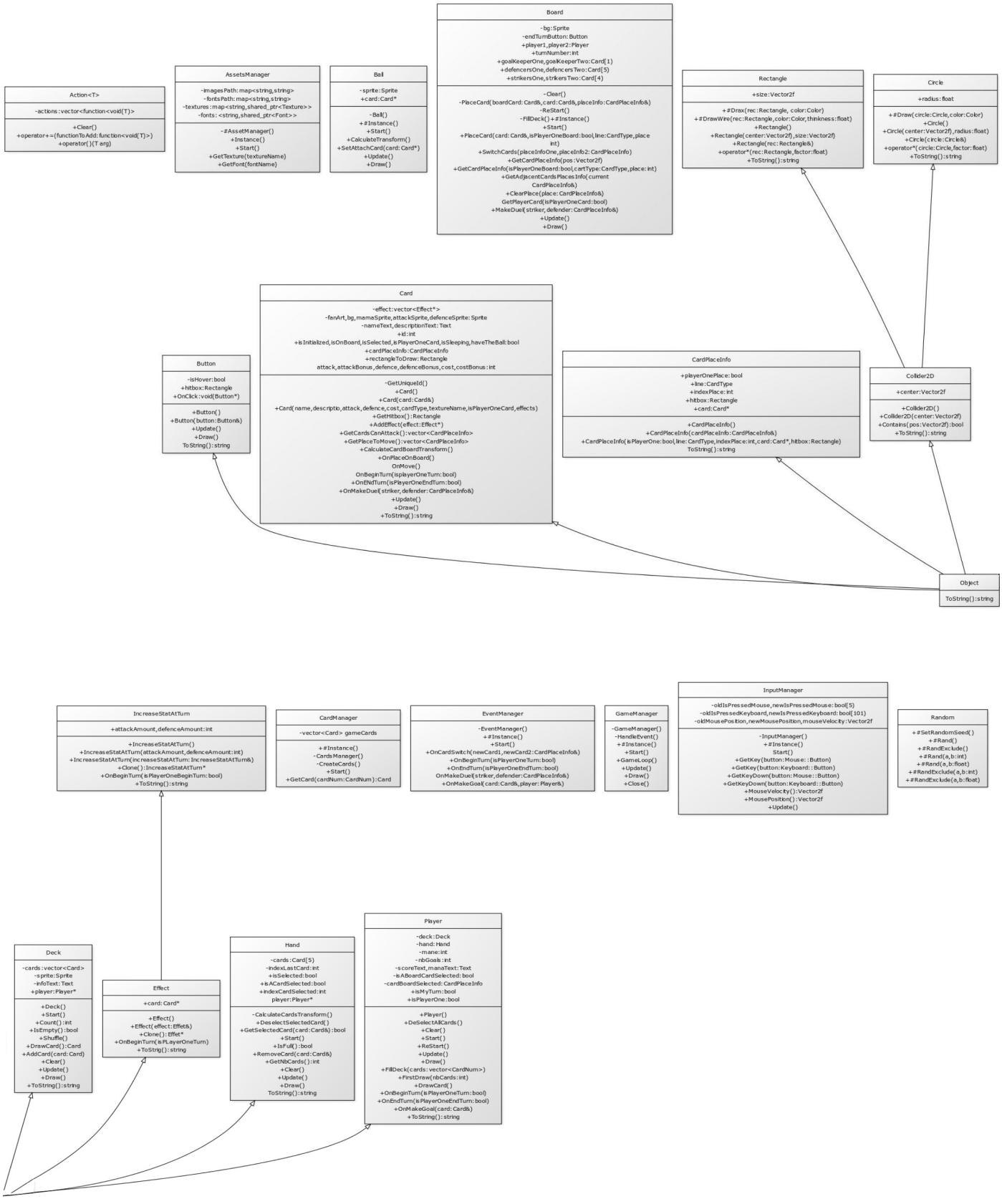
Noté que le nombre de cartes que possède votre adversaire est visible en haut de la fenêtre de jeu, bien évidemment les informations concernant la carte sont masquées.

Malheureusement il n'y a pas d'interface graphique permettant de créer son propre deck, cependant les deck joué par les 2 joueurs sont identiques et composé de quatre exemplaires des quatre cartes disponibles. Vous pouvez changer dans le code le deck des joueurs dans la fonction FillDeck dans le fichier board.cpp.

Cartes disponible dans le jeu

Diagramme UML du projet :

Je précise que le diagramme UML a été volontairement été légèrement simplifié, les 2 énumérations (CardNum et CardType) ne sont pas marquées car elles sont très simples.
Les méthodes statiques sont affichées avec le symbole # (alors que la convention de l'UML exige qu'il soit souligné).
De même, certaines méthodes que j'ai jugées de "peu pertinentes" ne sont pas affichées. De plus la classe Array2D et Action<T1,T2> et Action<T1,T2,T3> ne sont pas marquées car je ne les ai pas utilisées finalement. De même la classe Useful n'est pas marqué
car elle ne possède que des méthodes statique très basique.


Action<T>
-actions:vector<function<void(T)>
+Clear()
+operator+=(functionToAdd:function<void(T)>)
+operator()(T arg)
AssetsManager
-imagesPath:map<string,string>
-fontsPath:map<string,string>
-textures:map<string,shared_ptr<Texture>>
-fonts:<string,shared_ptr<Font>>
-#AssetManager()
+Instance()
+Start()
+GetTexture(textureName)
+GetFont(fontName)
Ball
-sprite:Sprite
+card:Card*
-Ball()
+#Instance()
+Start()
+CalculateTransform()
+SetAttachCard(card:Card*)
+Update()
+Draw()
Board
-bg:Sprite
-endTurnButton:Button
+player1,player2:Player
+turnNumber:int
+goalKeeperOne,goalKeeperTwo:Card[1]
+defencersOne,defencersTwo:Card[5]
+strikersOne,strikersTwo:Card[4]
-Clear()
-PlaceCard(boardCard:Card&,card:Card&,placeInfo:CardPlaceInfo&)
-ReStart()
-FillDeck()+#Instance()
+Start()
+PlaceCard(card:Card&,isPlayerOneBoard:bool,line:CardType,place int)
+SwitchCards(placeInfoOne,placeInfo2:CardPlaceInfo)
+GetCardPlaceInfo(pos:Vector2f)
+GetCardPlaceInfo(isPlayerOneBoard:bool,cartType:CardType,place:int)
+GetAdjacentCardsPlacesInfo(current CardPlaceInfo&)
+ClearPlace(place:CardPlaceInfo&)
GetPlayerCard(isPlayerOneCard:bool)
+MakeDuel(striker,defender:CardPlaceInfo&)
+Update()
+Draw()
Rectangle
+size:Vector2f
+#Drax(rec:Rectangle, color:Color)
+#DrawWire(rec:Rectangle,color:Color,thinkness:float)
+Rectangle()
+Rectangle(center:Vector2f),size:Vector2f)
+Rectangle(rec:Rectangle&)
+operator*(rec:Rectangle,factor:float)
+ToString():string
Circle
+radius:float
+#Draw(circle:Circle,color:Color)
+Circle()
+Circle(center:Vector2f),radius:float)
+Circle(circle:Circle&)
+operator*(circle:Circle,factor:float)
+ToString():string
Card
-effect:vector<Effect*>
-fanArt,bg,mamaSprite,attackSprite,defenceSprite:Sprite
-nameText,descriptionText:Text
+id:int
+isInitialized,isOnBoard,isSelected,isPlayerOneCard,isSleeping,haveTheBall:bool
+cardPlaceInfo:CardPlaceInfo
+rectangleToDraw:Rectangle
attack,attackBonus,defence,defenceBonus,cost,costBonus:int
-GetUniqueId()
+Card()
+Card(card:Card&)
+Card(name,descriptio,attack,defence,cost,cardType,textureName,isPlayerOneCard,effects)
+GetHitbox():Rectangle
+AddEffect(effect:Effect*)
+GetCardsCanAttack():vector<CardPlaceInfo>
+GetPlaceToMove():vector<CardPlaceInfo>
+CalculateCardBoardTransform()
+OnPlaceOnBoard()
OnMove()
OnBeginTurn(isplayerOneTurn:bool)
+OnEndTurn(isPlayerOneEndTurn:bool)
+OnMakeDuel(striker,defender:CardPlaceInfo&)
+Update()
+Draw()
+ToString():string
Button
-isHover:bool
+hitbox:Rectangle
+OnClick:void(Button*)
+Button()
+Button(button:Button&)
+Update()
+Draw()
ToString():string
CardPlaceInfo
+playerOnePlace:bool
+line:CardType
+indexPlace:int
+hitbox:Rectangle
+card:Card*
+CardPlaceInfo()
+CardPlaceInfo(cardPlaceInfo:CardPlaceInfo&)
+CardPlaceInfo(isPlayerOne:bool,line:CardType,indexPlace:int,card:Card*,hitbox:Rectangle)
ToString():string
Collider2D
+center:Vector2f
+Collider2D()
+Collider2D(center:Vector2f)
+Contains(pos:Vector2f):bool
+ToString():string
Object
ToString():string
IncreaseStatAtTurn
+attackAmount,defenceAmount:int
+IncreaseStatAtTurn()
+IncreaseStatAtTurn(attackAmount,defenceAmount:int)
+IncreaseStatAtTurn(increaseStatAtTurn:IncreaseStatAtTurn&)
+Clone():IncreaseStatAtTurn*
+OnBeginTurn(isPlayerOneBeginTurn:bool)
+ToString():string
CardManager
-vector<Card> gameCards
+#Instance()
-CardsManager()
-CreateCards()
+Start()
+GetCard(cardNum:CardNum):Card
EventManager
-EventManager()
+#Instance()
+Start()
+OnCardSwitch(newCard1,newCard2:CardPlaceInfo&)
+OnBeginTurn(isPlayerOneTurn:bool)
+OnEndTurn(isPlayerOneEndTurn:bool)
OnMakeDuel(striker,defender:CardPlaceInfo&)
+OnMakeGoal(card:Card&,player:Player&)
GameManager
-GameManager()
-HandleEvent()
+#Instance()
+Start()
+GameLoop()
+Update()
+Draw()
+Close()
InputManager
-oldIsPressedMouse,newIsPressedMouse:bool[5]
-oldIsPressedKeyboard,newIsPressedKeyboard:bool[101]
-oldMousePostion,newMousePosition,mouseVelocity:Vector2f
-InputManager()
+#Instance()
Start()
+GetKey(button:Mouse::Button)
+GetKey(button:Keyboard::Button)
+GetKeyDown(button:Mouse::Button)
+GetKeyDown(button:Keyboard::Button)
+MouseVelocity():Vector2f
+MousePosition():Vector2f
+Update()
Random
+#SetRandomSeed()
+#Rand()
+#RandExclude()
+#Rand(a,b:int)
+#Rand(a,b:float)
+#RandExclude(a,b:int)
+#RandExclude(a,b:float)
Deck
-cards:vector<Card>
-sprite:Sprite
-infoText:Text
+player:Player*
+Deck()
+Start()
+Count():int
+IsEmpty():bool
+Shuffle()
+DrawCard():Card
+AddCard(card:Card)
+Clear()
+Update()
+Draw()
+ToString():string
Effect
+card:Card*
+Effect()
+Effect(effect:Effet&)
+Clone():Effet*
+OnBeginTurn(isPlayerOneTurn)
+ToStrig():string
Hand
-cards:Card[5]
-indexLastCard:int
+isSelected:bool
+isACardSelected:bool
+indexCardSelected:int
player:Player*
-CalculateCardsTransform()
+DeselectSelectedCard()
+GetSeleectedCard(card:Card&):bool
+Start()
+IsFull():bool
+RemoveCard(card:Card&)
+GetNbCards():int
+Clear()
+Update()
+Draw()
ToString():string
Player
-deck:Deck
-hand:Hand
-mana:int
-nbGoals:int
-scoreText,manaText:Text
-isABoardCardSelected:bool
-cardBoardSelected:CardPlaceInfo
+isMyTurn:bool
+isPlayerOne:bool
+Player()
+DeSelectAllCards()
+Clear()
+Start()
+ReStart()
+Update()
+Draw()
+FillDeck(cards:vector<CardNum>)
+FirstDraw(nbCards:int)
+DrawCard()
+OnBeginTurn(isPlayerOneTurn:bool)
+OnEndTurn(isPlayerOneEndTurn:bool)
+OnMakeGoal(card:Card&)
+ToString():string

Procédure d’installation

Le jeu utilise la SFML en version x64, ainsi il n'est pas possible d'exécuter le jeu sur une machine 32bit.
Il a été développé en C++ sous l’IDE visual studio 2022 avec le compilateur et débugueur microsoft visual C++.

Windows :
Pour lancer le jeu vous pouvez directement télécharger le dossier avec le code compilé avec microsoft visual C++ se trouvant dans : x64/Release puis lancer l'exécutable PaniniGameCard.exe.

Si vous voulez recompiler le programme et/ou modifier le code il vous faudra visual studio 2022 avec le package développement C++. Puis ouvrez le projet en lançant Panini GameCard.sln. Ensuite pour configurer les dépendances il faut allé dans les propriété du projet, propriété de configuration => Répertoire VC++ et dans la case répertoires Include et répertoire include public sélectionner le fichier include du répo. De même pour pour le répertoire de bibliothèques sélectionner le dossier lib.
Le projet est maintenant configuré, pour pouvez recompiler et modifier le code source.
Attention les dll du dossier bin sont à copié dans le dossier de l'exécutable si vous souhaitez redistribuer le jeu.

Linux :
Pour lancer le jeu sur linux télécharger les dossiers Assets, include, lib_linux ainsi que l’ensemble des .cpp et .hpp et le makefile du répertoire GitHub.
La commande make permet de créer l'exécutable du jeu (PaniniCardGame). La commande make clean permet de supprimer l’exécutable ainsi que les .o inutiles.

Partie de l’implémentation dont je suis le plus fier

La classe AssetManager est vraiment un moyen simple et optimisé pour charger les assets utilisé par notre jeu.
En effet il suffit pour charger une asset dans le jeu (comme une image ou une police de caractère) de ce rendre dans AssetManager.hpp puis ajouter une ligne supplémentaire dans le dictionnaire approprié :
Une clef pour pouvoir identifier l'asset de manière unique ainsi que le chemin d'accès à cette dernière.
La classe se chargera toute seule de chargé l'asset en mémoire.
Comme AssetManager est un singleton, on peut y accéder n'importe où dans le code et récupérer une référence de l'asset voulu en appelant la méthode AssetManager::Instance().GetTexture() ou GetFont ect...

De même, la classe InputManager permet très facilement de récupérer les entrées du clavier et de la souris. Cette classe est également un singleton ainsi nous pouvons accéder à l'unique instance de InputManager en tapant la ligne InputManager::Instance().
On peut y récupérer la position de la souris par rapport au coin haut gauche de la fenêtre de jeu avec la méthode MousePosition(). On peut aussi récupérer la vitesse de la souris avec MouseVelocity(). De plus les méthode GetKey() prennent en entrée un bouton du clavier ou de la souris et renvoie true si le bouton est appuyé lors de l'image courante, false sinon. De même pour la méthode GetKeyDown() mais qui ne renverra true lors d'une seule image, au moment de l'appuie sur la touche.

De plus, la classe Card est assez complète et simple à utiliser. L'idée de mettre des méthode dit "d'évent" qui se lance à des moment précis du jeu (comme la méthode OnMove ou OnBeginTurn etc) permettent d'ajouter facilement des effets.
En effet les effets de cartes doivent s'activer selon des conditions assez précise, ainsi appelé des fonction d'événement à chaque action du jeu rend l'implémentation d'effet plus simple.