



Element Wars

RPG type Pokémon en C++

ARNOLL	Christopher
BOUNYASIT	Vibert
DARWANE	Ilies
NSHIMIYIMANA	Robert

Tuteur : TRAHAY François

27/05/2015

Soutenance de fin de CSC3502



Objectifs du projet

- S'initier au C++ et la PO (Robert & Christopher)
- Programmer en équipe
- Réaliser un jeu de rôle avec système de combat au tour par tour



Plan de la présentation

- Introduction
- Détails du cahier des charges
- Problèmes rencontrés
- Projet final



Introduction

- Qu'est ce qu'un RPG ?
- Le projet dans les grandes lignes

■ Définition

■ Notre projet:

- Mode déplacement
- Mode Combat

« Une Image vaut mieux qu'un long discours »



Introduction

Qu'est ce qu'un RPG ?

« Une Image vaut mieux qu'un long discours »





Détails du cahier des charges

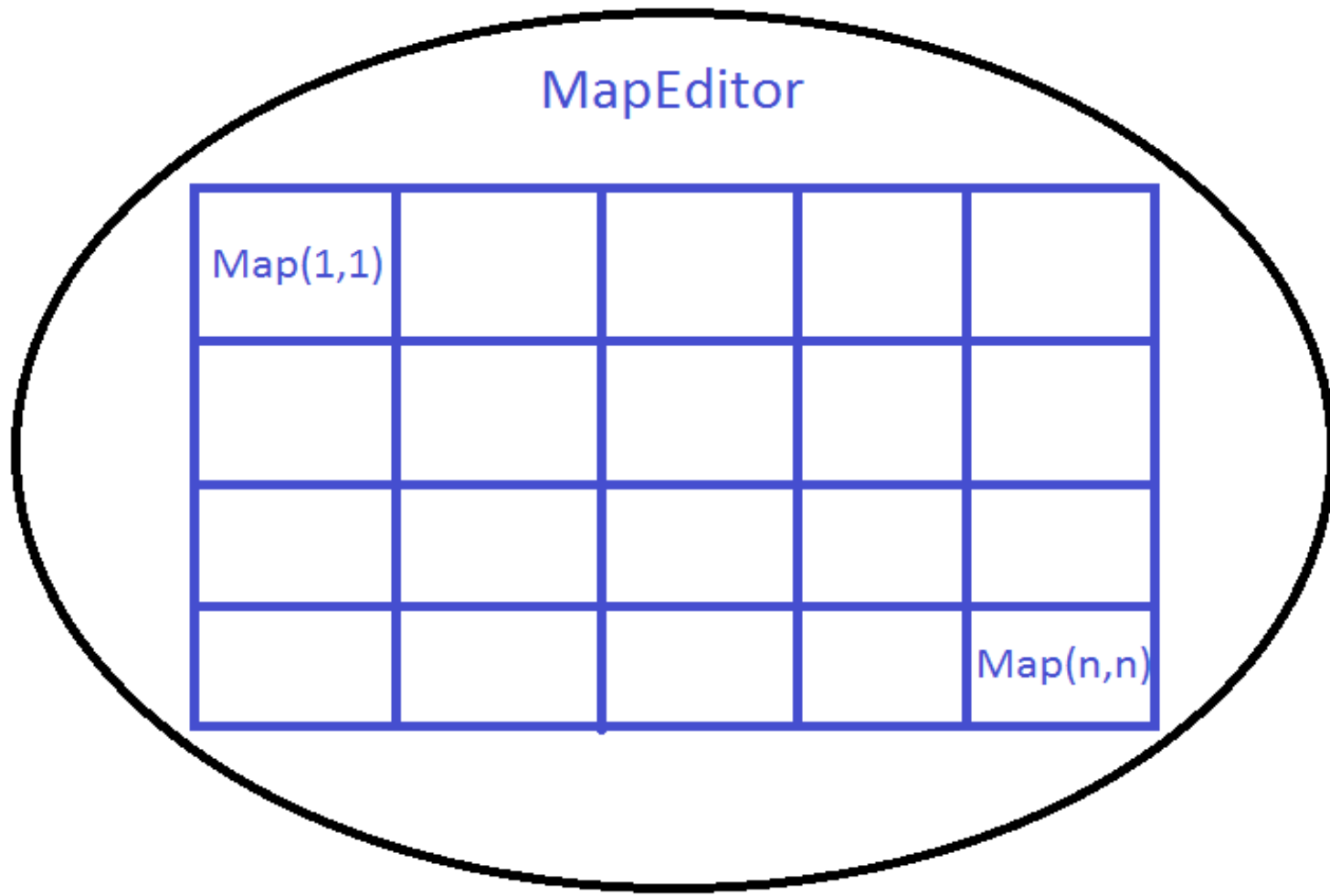
- **Partie graphique**
- **Partie physique**
- **Partie combat**
- **Partie menu**
- **Partie sonore**



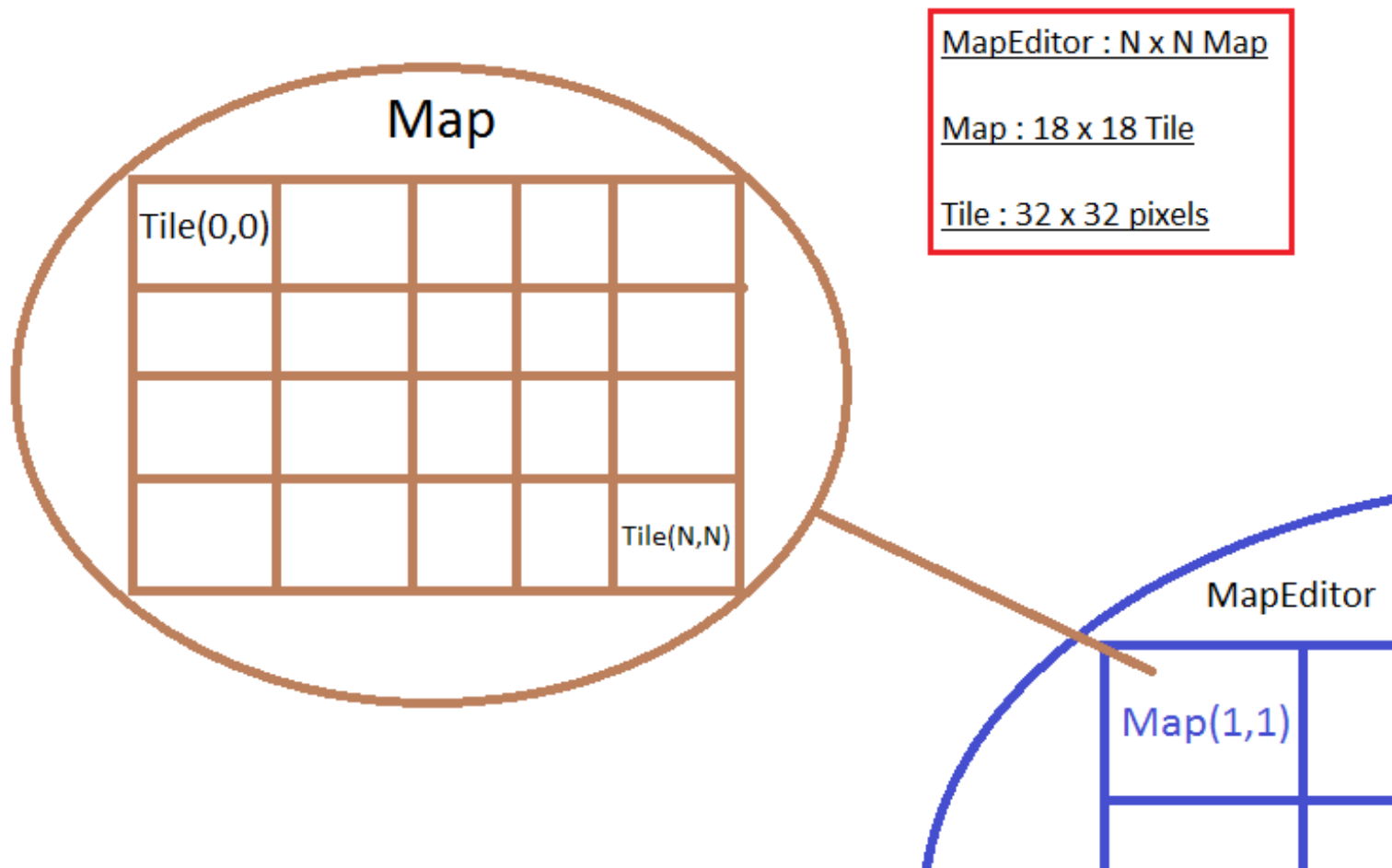
Partie graphique



Objet MapEditor



Objet Map



Objet Tile

Tile

x, y : Integer

graphicParam : GraphicParam

physicParam : PhysicParam

linkId : Integer

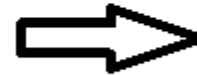
secondObject, thirdObject : Integer

Objet Tile

Tile

x, y : Integer

graphicParam : GraphicParam



index du Sprite à charger
(image à placer sur la case
de coordonnée (x,y))

physicParam : PhysicParam

linkId : Integer

secondObject, thirdObject : Integer

Objet Tile

Tile

x, y : Integer

graphicParam : GraphicParam

physicParam : PhysicParam



paramètre physique de la
case de coordonnée (x,y)

- 0 - non traversable
- 1 - traversable
- 2 - affecté à un NPC
- 3 - changement de map
(bâtiments)
- 4 - vitesse accéléré
- 5 - vitesse ralentie
- 6 - zone de combat

linkId : Integer

secondObject, thirdObject : Integer

Objet Tile

Tile

x, y : Integer

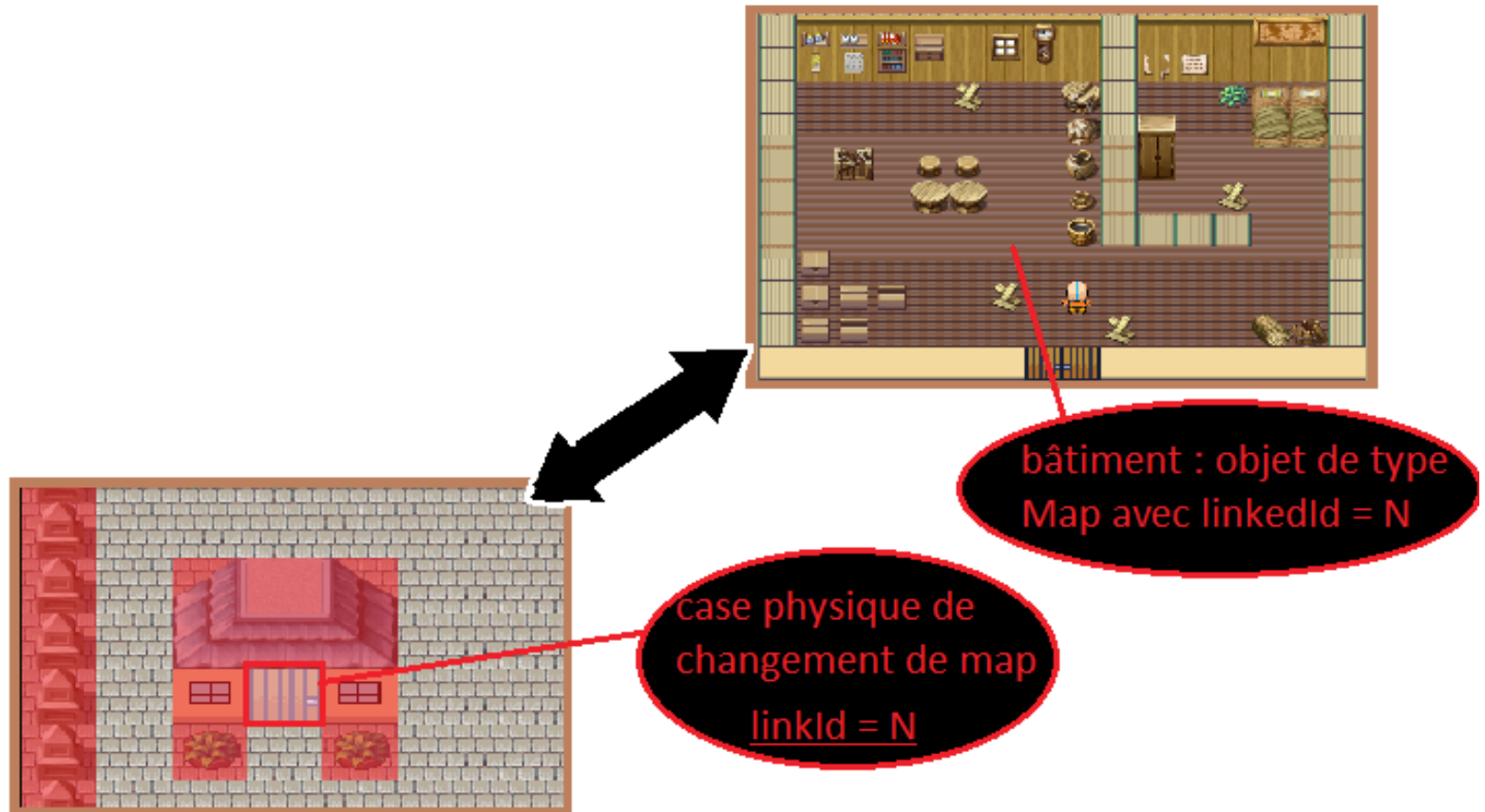
graphicParam : GraphicParam

physicParam : PhysicParam

linkId : Integer

secondObject, thirdObject : Integer

Explication de linkId et linkedId



Objet Tile

Tile

x, y : Integer

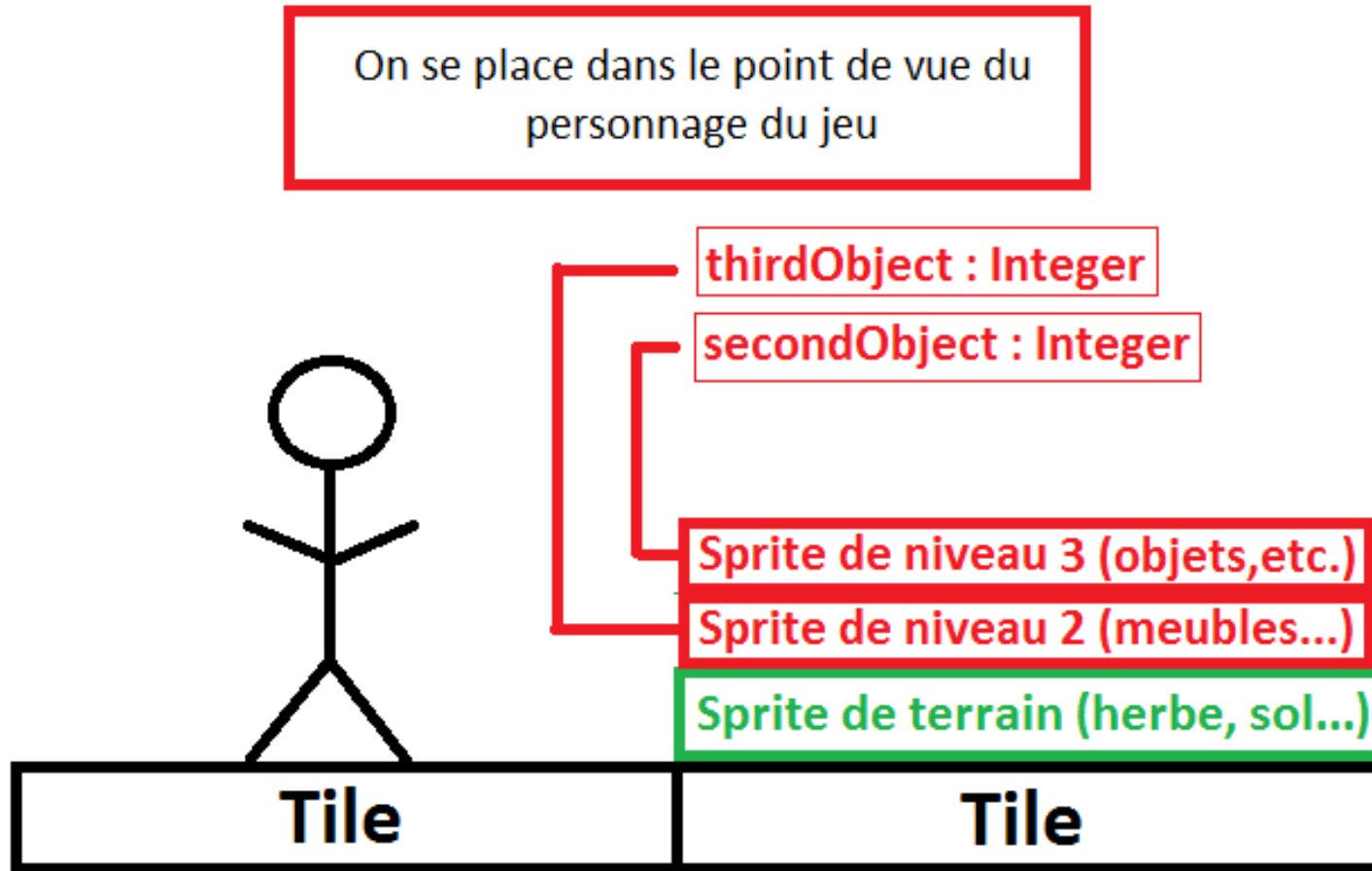
graphicParam : GraphicParam

physicParam : PhysicParam

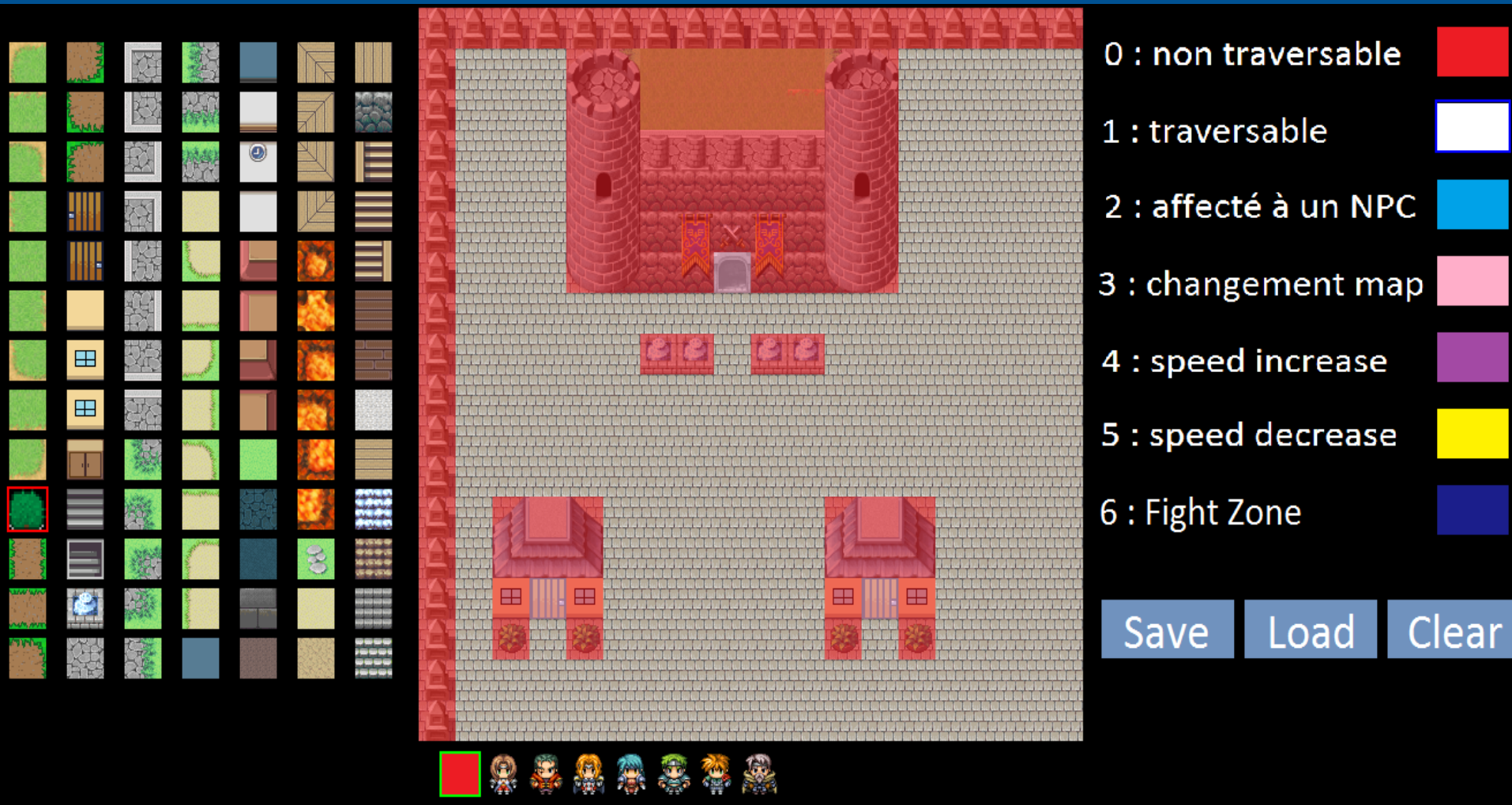
linkId : Integer

secondObject, thirdObject : Integer

Modélisation de secondObject/thirdObject



Interface de l'éditeur



The image displays a game editor interface. On the left is a large grid of various tiles, including grass, stone, wood, and special effects like fire and speed changes. The central area shows a preview of a map with a castle, two towers, and some buildings. On the right is a legend with six entries, each with a colored square and a description. At the bottom right are three buttons: 'Save', 'Load', and 'Clear'. At the bottom left, there is a row of character icons and a red square.

0 : non traversable

1 : traversable

2 : affecté à un NPC

3 : changement map

4 : speed increase

5 : speed decrease

6 : Fight Zone

Save Load Clear



Objet MapEditor

MapEditor

```
currentMap : Map  
currentGraphicId : Integer  
currentObjectId : Integer  
currentPhysicId : Integer  
currentNpcId : Integer  
currentSpriteTab : Integer
```

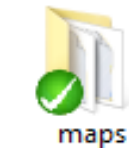
Lancement du programme

Etapes du lancement de MapEditor

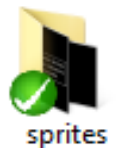
```
"C:\Users\vayken\Desktop\projet C\ProjetC\bin\Debug\ProjetC
Initiation de la phase de chargement de map.
Veuillez choisir la coordonnee X de la map a charger :
1
Veuillez choisir la coordonnee Y de la map a charger :
1
```



```
MapEditor editor(refWin, x, y);
editor.run();
```



maps



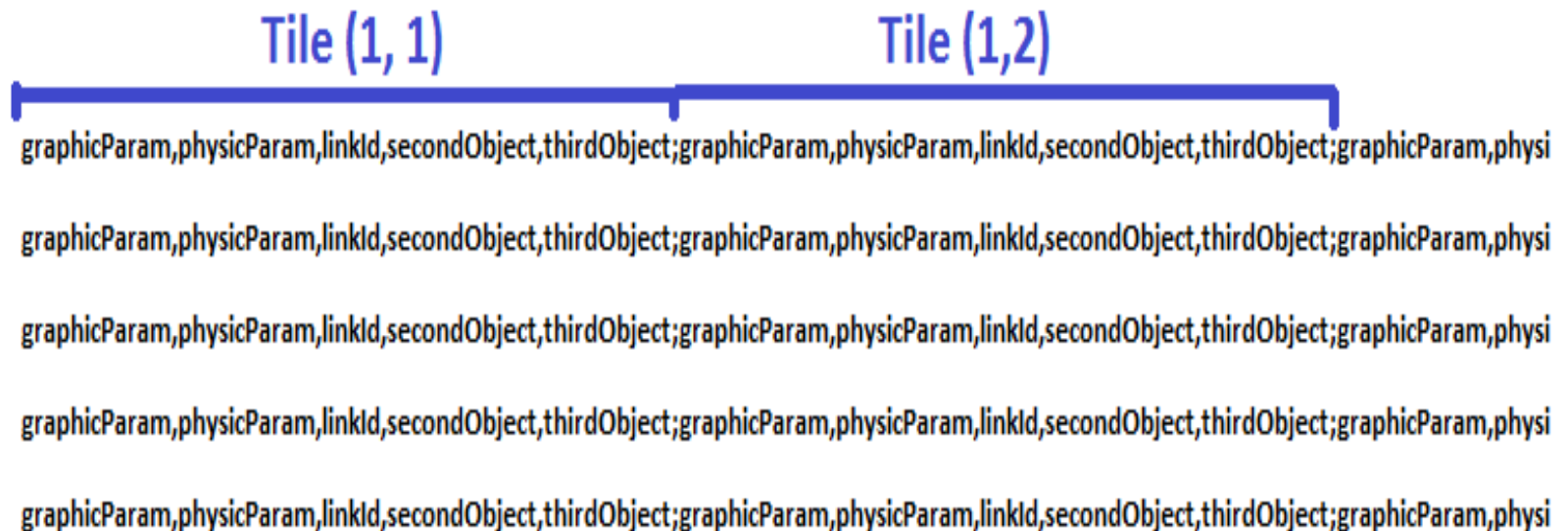
sprites

écran

```
m_gameWindow.create(sf::VideoMode(WINDOWS_FINAL_SI
loadConfig(x, y);
loadTextures();
update();
cout << "Chargement termine !" << endl;
```



Contenu d'un fichier Map .txt





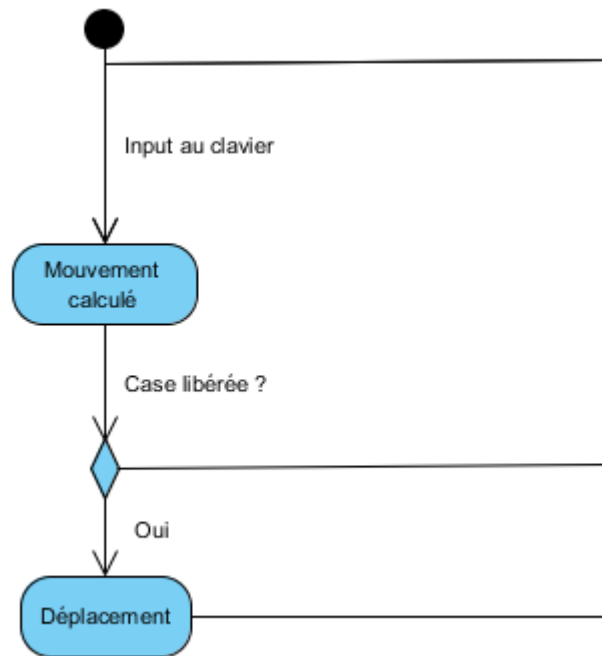
Partie physique



Détails du cahier des charges

Partie physique

Représentation simplifiée de la boucle de mouvement



Commençons à détailler:

- Mouvement unidirectionnel
- L'input précédent est conservé.

Détails du cahier des charges

Partie physique

■ Calcul du mouvement

```
velocity = movement*deltaTime.restart().asSeconds();
```

Détails du cahier des charges

Partie physique

■ Calcul du mouvement

```
velocity = movement * deltaTime.restart().asSeconds() ;
```

Vecteur
vitesse

Vecteur
accélération

Temps

Concrètement ici, on récupère la valeur et on
remet l'horloge à zéro

Détails du cahier des charges

Partie physique

■ Calcul du mouvement

```
velocity = movement * deltaTime.restart().asSeconds() ;
```

Vecteur
vitesse

Vecteur
accélération

Temps

Concrètement ici, on récupère la valeur et on
remet l'horloge à zéro

Le mouvement par accélération permet d'obtenir un mouvement fluide
quelque soit le processeur.

On stock la vitesse pour la gestion de fin de mouvement.

Détails du cahier des charges

Partie physique

■ Gestion de fin de mouvement

```
if(m_movement==1){
    caseToReach = (int) (position.y)/32.f;
    pos.x = position.x;
    pos.y = caseToReach*32;
    if(!isTraversable(pos))
        tv = false;

    Yrel = caseToReach*32.f;

    while(m_gameWindow.isOpen() && position.y != Yrel && tv ){
        processEvents();
        position += m_lastSpeed;
        if(Yrel - position.y > 0)
            position.y = Yrel;

        m_animationCount++;
        if(m_animationCount>20) m_animationCount=1;
        m_Player.setTexture(m_playerTextures[m_animationCount / 7 + 9]);
        m_Player.setPosition(position);
        borderControl();
        updateBackground();
        render();
    }
    m_animationCount=0;
    m_Player.setTexture(m_playerTextures[10]);
}
```

Calcul de la distance à parcourir

Détails du cahier des charges

Partie physique

■ Gestion de fin de mouvement

```
if(m_movement==1){
    caseToReach = (int) (position.y)/32.f;
    pos.x = position.x;
    pos.y = caseToReach*32;
    if(!isTraversable(pos))
        tv = false;

    Yrel = caseToReach*32.f;

    while(m_gameWindow.isOpen() && position.y != Yrel && tv ){
        processEvents();
        position += m_lastSpeed;
        if(Yrel - position.y > 0)
            position.y = Yrel;

        m_animationCount++;
        if(m_animationCount>20) m_animationCount=1;
        m_Player.setTexture(m_playerTextures[m_animationCount / 7 + 9]);
        m_Player.setPosition(position);
        borderControl();
        updateBackground();
        render();
    }
    m_animationCount=0;
    m_Player.setTexture(m_playerTextures[10]);
}
```

Calcul de la distance à parcourir

Boucle de mouvement

Détails du cahier des charges

Partie physique

■ Gestion de fin de mouvement

```
if(m_movement==1){
    caseToReach = (int) (position.y)/32.f;
    pos.x = position.x;
    pos.y = caseToReach*32;
    if(!isTraversable(pos))
        tv = false;

    Yrel = caseToReach*32.f;

    while(m_gameWindow.isOpen() && position.y != Yrel && tv ){
        processEvents();
        position += m_lastSpeed;
        if(Yrel - position.y > 0)
            position.y = Yrel;

        m_animationCount++;
        if(m_animationCount>20) m_animationCount=1;
        m_Player.setTexture(m_playerTextures[m_animationCount / 7 + 9]);
        m_Player.setPosition(position);
        borderControl();
        updateBackground();
        render();
    }
    m_animationCount=0;
    m_Player.setTexture(m_playerTextures[10]);
}
```

Calcul de la distance à parcourir

Boucle de mouvement

On s'assure que le personnage reste aligné sur la « grille » de la Map

Détails du cahier des charges

Partie physique

■ Gestion des collisions



```
bool Engine::isTraversable(sf::Vector2f pos){
    int currentTileX_top, currentTileY_top, currentTileX_bot, currentTileY_bot;
    currentTileX_top = (pos.x)/32;
    currentTileY_top = (pos.y)/32;
    currentTileX_bot = (pos.x+23)/32;
    currentTileY_bot = (pos.y+31)/32;
    if(m_currentMap.getTile(currentTileY_top, currentTileX_top).getPhysicParam().getPhysicId() == 0 ||
        return false;
    }
    else if(m_currentMap.getTile(currentTileY_bot, currentTileX_bot).getPhysicParam().getPhysicId() == 0
        return false;
    }
    return true;
}
```

Détection des obstacles par analyse de la position des coins haut gauche et bas droit du personnage

■ Gestion des transitions (bords et bâtiments)

Sur le même principe que précédemment, on analyse la position des coins opposés.

On charge alors la Map (ou « sous-Map » pour un bâtiment) adéquate. On réinitialise la conservation cinétique. On positionne le personnage à la position adéquate.

Détails du cahier des charges

Partie physique

■ Gestion des transitions (bords et bâtiments)

Sur le même principe que précédemment, on analyse la position des coins opposés.

On charge alors la Map (ou « sous-Map » pour un bâtiment) adéquate. On réinitialise la conservation cinétique. On positionne le personnage à la position adéquate.



```
/****** ENTREE DANS UN BATIMENT *****/
if(m_currentMap.getTile(currentTileY_top,currentTileX_top).getPhysicParam().getPhysicId() == 3){
    int id;
    id = m_currentMap.getTile(currentTileY_top,currentTileX_top).getLinkId();
    if(id!=-1){
        loadConfig(m_mapX, m_mapY,-1);
        position = savedPosition;
        m_Player.setPosition(position.x,position.y);
        m_lastSpeed.x=0.f;
        m_lastSpeed.y=0.f;
        return;
    }
    m_lastSpeed.x=0.f;
    m_lastSpeed.y=0.f;
    loadConfig(m_mapX, m_mapY,id);
    savedPosition = position;
    position.x = 9*32.f;
    position.y = 16*32.f;
    return;
}
```

Détails du cahier des charges

Partie physique

■ Gestion des animations



0



3



6



9

3 images par « mouvement »

Détails du cahier des charges

Partie physique

■ Gestion des animations



0



3



6



9

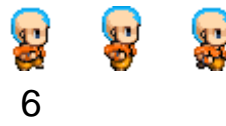
3 images par « mouvement »

Après expérimentation, avec un verrouillage logiciel (volontaire) de 60 images affichées par seconde, un mouvement se réalise en 22 images. Donc on change la texture affichée par la texture adéquate toutes les 7 images affichées.

Détails du cahier des charges

Partie physique

■ Gestion des animations



3 images par « mouvement »

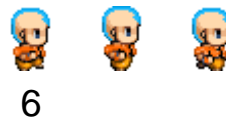
Après expérimentation, avec un verrouillage logiciel (volontaire) de 60 images affichées par seconde, un mouvement se réalise en 22 images. Donc on change la texture affichée par la texture adéquate toutes les 7 images affichées.

```
m_animationCount++;  
if(m_animationCount>20) m_animationCount=1;  
m_Player.setTexture(m_playerTextures[m_animationCount / 7 + 9]);  
m_Player.setPosition(position);
```

Détails du cahier des charges

Partie physique

■ Gestion des animations



3 images par « mouvement »

Après expérimentation, avec un verrouillage logiciel (volontaire) de 60 images affichées par seconde, un mouvement se réalise en 22 images. Donc on change la texture affichée par la texture adéquate toutes les 7 images affichées.

```
m_animationCount++;  
if(m_animationCount>20) m_animationCount=1;  
m_Player.setTexture(m_playerTextures[m_animationCount / 7 + 9]);  
m_Player.setPosition(position);
```

↓
Décalage indiciel

Détails du cahier des charges

Partie physique

■ Gestion des combats

Lorsque le personnage marche sur une case où un combat peut se produire, on génère un nombre entre 0 et 100 qu'on additionne à un compteur en mémoire. Lorsque ce compteur dépasse 8000 (un combat toutes les 9-12 secondes), on déclenche le combat.

Détails du cahier des charges

Partie physique

■ Gestion des combats

Lorsque le personnage marche sur une case où un combat peut se produire, on génère un nombre entre 0 et 100 qu'on additionne à un compteur en mémoire. Lorsque ce compteur dépasse 8000 (un combat toutes les 9-12 secondes), on déclenche le combat.

```
void Engine::fightController(sf::Clock deltaTime){
    int currentTileX_top, currentTileY_top, currentTileX_bot, currentTileY_bot;
    currentTileX_top = (position.x)/32;
    currentTileY_top = (position.y)/32;
    currentTileX_bot = (position.x+23)/32;
    currentTileY_bot = (position.y+31)/32;
    if(m_currentMap.getTile(currentTileY_top, currentTileX_top).getPhysicParam().getPhysicId() == 6 && m_movement != 0){
        randFight += rand() % 100;

    }
    else if(m_currentMap.getTile(currentTileY_bot, currentTileX_bot).getPhysicParam().getPhysicId() == 6 && m_movement != 0){
        randFight += rand() % 100;
    }

    if(randFight > 8000){
        endAnimation(deltaTime);
        // transition();
        music.pause();
        Combat combat(m_gameWindow);
        music.play();
        randFight=0;
    }
}
```



Partie combat





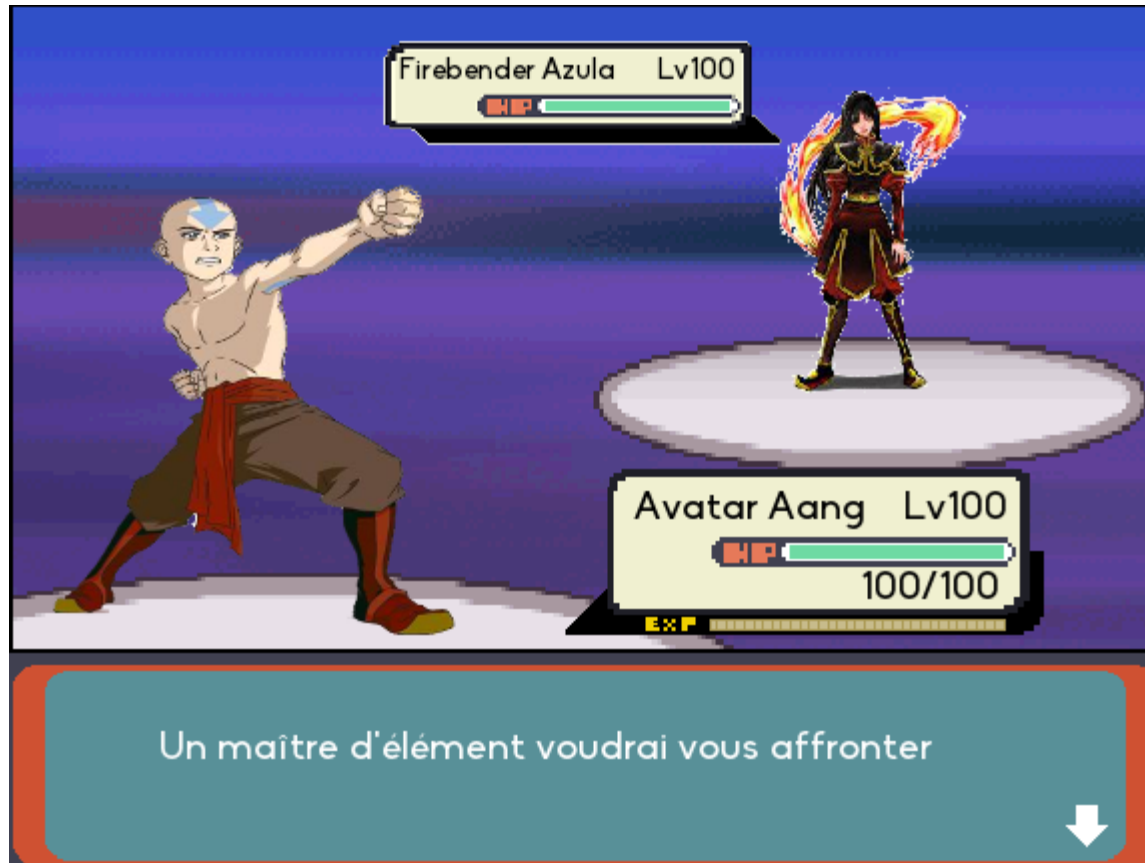
Combat

- Organisé en différent « Etats » de combat
- L'écran se dessine en fonction de l'« état » dans lequel se trouve l'utilisateur

Chargement des données de combat :

- **Objet CombatData** qui regroupe les stats de combat(points de vies, niveau, attaque, defense, etc...)
- **Personnages à charger** pour ce combat(allié et ennemi)
- **Placement de l'utilisateur à l'état 0 de combat**, c'est-à-dire l'entrée en combat
- **Dessin de l'interface de combat**

Interface de combat



L'interface inférieure change en fonction de l'état de combat



Etats de combat (interface)

Un maître d'élément voudrai vous affronter



Quelle action va faire Avatar Aang	+Attaquer Changer Objet Fuir
---------------------------------------	-------------------------------------



Air X → Tornade X	Armure X Soin X
----------------------	--------------------

Etape 1

Etape 2

Etape 3



Partie Menu



Détails du cahier des charges

Partie Menu





Partie Sonore



Implémentation du son

Choix stratégiques

- **Bruitages sonore**
- **Musiques d'ambiance**

■ Moment de la démonstration

Conclusions

- **Ce que l'on a fait:**
 - Partie graphique,
 - Partie physique,
 - Partie combat,
 - Partie menu
 - Partie sonore.
- **Ce que l'on voudrait ajouter:**
 - un scénario,
 - les NPC
 - + de sons
- **Ce que l'on a appris**

Place à vos questions

