**Convention de nommage Halite en C++**

**GENERAL**

**Utiliser des tabulations de 4 espaces.**

**Pas de documentation, juste explication du code quand nécessaire**

**Utiliser des accolades partout (if, else, functions, structures, typedefs, class definitions, etc.)**

if (x) {

}

**La déclaration du else est sur la même ligne que la dernière accolade fermante du if.**

if (x) {

} else {

}

**Ne pas mettre d’espace dans les expressions entre parenthèses**

if (x) { et x = (y \* 0.5f);

}

A la place de

if ( x ) { et x = ( y \* 0.5f );

}

**Utilisez la spécification de précision pour les valeurs à virgule flottante à moins qu'il n'y ait un besoin explicite d'un double.**

float f = 0.5f; et float f = 1.0f;

A la place de

float f = 0.5; et float f = 1.f;

**Les noms de fonction commencent par une majuscule et chaque mot suivant par une majuscule.**

void CamelCase();

**Utiliser des références sur des constantes en paramètres de fonction si ces paramètres ne seront pas modifiés. Les paramètres ont la même convention de nommage que des variables.**

int GetID(const Entity& kr\_entity);

**Les noms de variables locale commencent avec une lettre minuscule et chaque mot suivant par une majuscule.**

float camelCase;

**Les noms des déclarations typedef commencent par une majuscule et finissent par « Type ».**

typedef std::string CommandType;

**Les noms des struct commencent par une majuscule.**

struct Dropoff;

**Utiliser des enum class au lieu des enum** [**(#différence**](https://stackoverflow.com/questions/18335861/why-is-enum-class-preferred-over-plain-enum)**), les noms des enum commencent par une majuscule, les constantes sont en majuscules, et spécifier le type de la valeur si besoin.**

enum class Direction : char {

NORTH = 'n',

…

}

**Les noms des fonctions récursives finissent par « Recursive** **».**

std::vector<MapCell> getSuccessorsRecursive(int n);

**Utiliser ‘const’ autant que possible.**

Utiliser :

const int\* p; // pointer to const int

int\* const p; // const pointer to int

const int\* const p; // const pointer to const int

Ne pas utiliser :

int const\* p;

**L’astérisque du pointeur ou le & de la référence sont collés au type.**

MapCell\* p\_cell;

MapCell& r\_cell;

**Si la variable est static, préfixer de « s ».**

static int s\_killCpt;

**Si la variable est constante, l’écrire en majuscule.**

const float k\_pi = 3.141592f;

**Si la variable est un pointeur, la préfixer de « p ».**

Player\* p\_playerArray = new Player[5];

**Si la variable est un pointeur de pointeur.**

Player\*\* pp\_playerArray = new Player[5][2];

**Si la variable est une reference.**

Player& r\_player(jhon);

**Ne pas mettre de using dans les hpp. Utiliser à la place std:: .**

**CLASSES**

**Struct si l’objet ne fait que contenir des données, sinon Class.**

**Préfixer les interfaces d’un « I ».**

class IShape {

…

};

**Préfixer les classes virtuelles pures par « Abstract ».**

class AbstractButton {

…

};

**Utiliser des namespace différent pour nos propres classes, regrouper par « bibliothèques ».**

namespace fsm { ou namespace nav {

… …

} }

**Séparer définition et déclaration d’une classe dans un fichier cpp et un fichier hpp.**

**Utiliser des #pragam once au lieu des #(if/n)define dans les hpp.**

**Le nom d’une classe commence par une majuscule.**

class GameMap {};

**Les noms des variables de classe (pas de Struct) respectent la même convention de nommage que les variables mais avec un « m » devant. Le « m » précède toujours les autres préfixes. Faire deux colognes.**

class Screen {

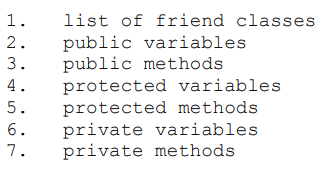
const int mk\_widthOfScreen = 600;

int\* mp\_heightOfScreen = new int(400);

static int ms\_connectedSceen = 0;

}

**L’arrangement des variables et des méthodes dans une classe suit l’ordre suivant :**

****

**Toujours marquer les méthodes « const » quand elles ne modifient aucune variable de la classe.**

void printElements() const;

**Retourner des « const » au maximum à part si l’usage de l’objet est d’être modifié.**

const ID getID() const;