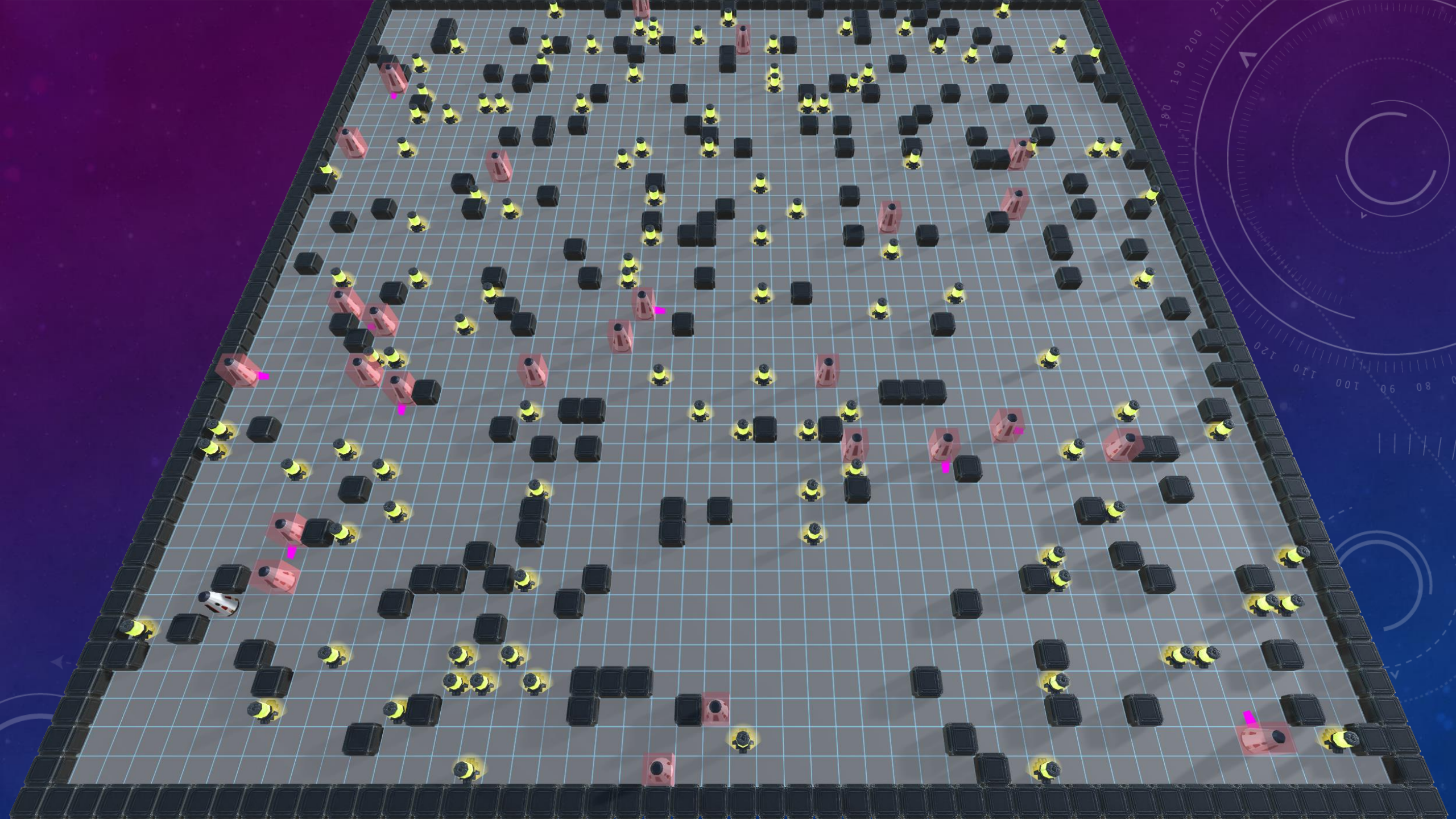
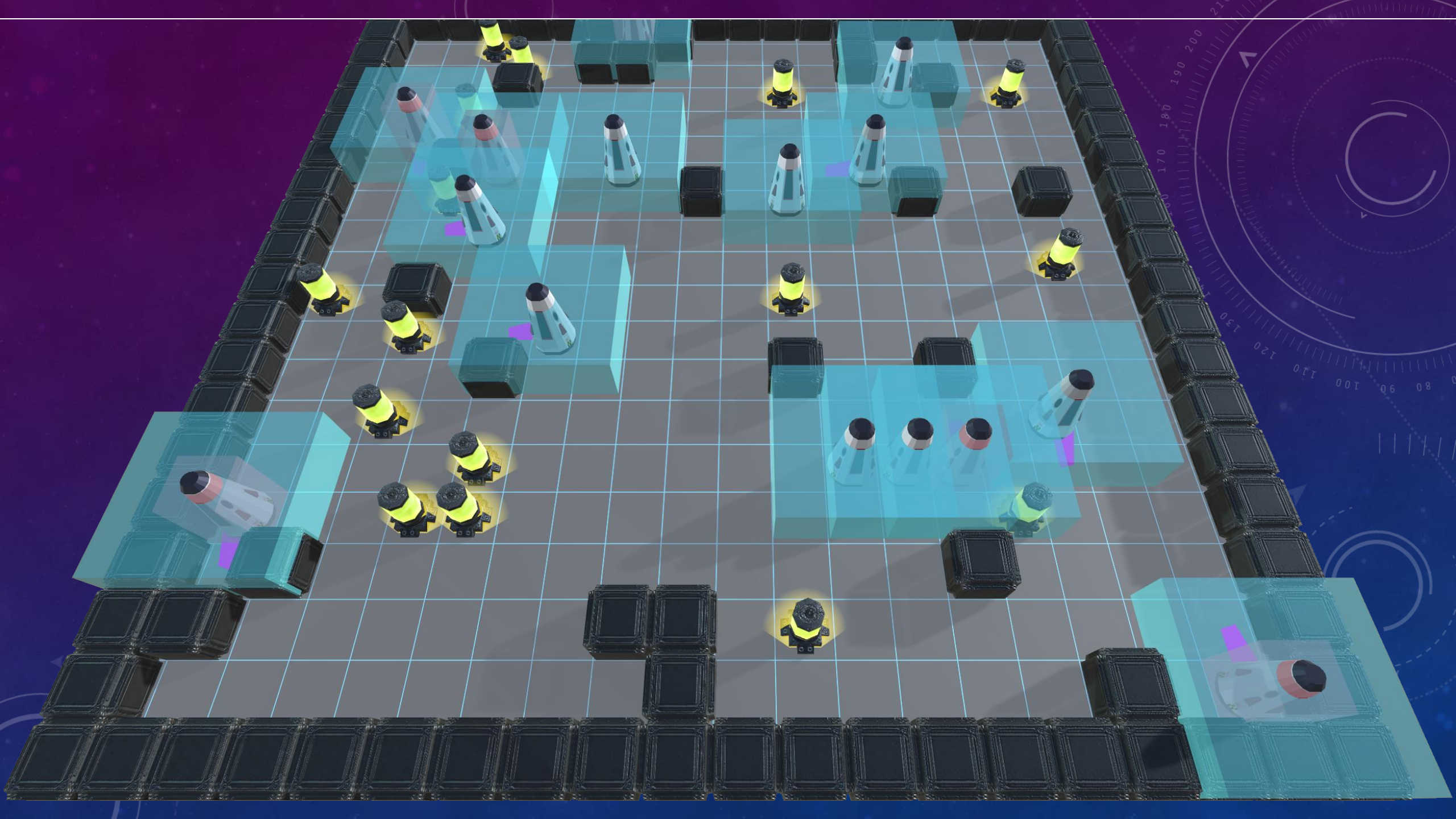
PROJET INFORMATIQUE
3ÈME ANNÉE

STÉPHANE FARDOUX

Battle IA





OBJECTIF DU PROJET

1. Coder des BOTs avec une I.A. en language C# et/ou Javascript
2. Coder un rendu graphique des informations de votre I.A. en html/css/Javascript

Il vous est fourni

- Le moteur d'exécution de la simulation
- Un rendu graphique de la simulation
- Un BOT exemple en C#

LA SIMULATION

Le moteur d'exécution de la simulation fonctionne en mode console.

Il affiche en format texte tout ce qu'il se passe.

Ce moteur supporte un nombre « infini » de Bots.

Vous pilotez la simulation via quelques commandes simples. (Touche G pour Go!)

C'est un serveur sur lequel vient se connecter un afficheur du terrain de simulation et les Bots.

La technologie utilisée : websocket.

Le protocole de communication est fournit.

ARCHITECTURE

Moteur d'exécution globale
multi-joueurs

Client pour
rendu global

I.A. à
créer : le BOT

Client pour afficher
les infos de l'IA : COCKPIT

TERRAIN DE SIMULATION

- Le terrain est constitué par des cases rectangulaires, il est entièrement bordé de cases.
- La superficie du terrain n’est pas connue par le BOT !
- Une BOT occupe 1 case.
- Une unité d’énergie occupe 1 case.
- Vous ne savez pas combien il y a de BOT dans la simulation.

1 TOUR DE JEU

Un tour de jeu consiste en une séquence de communication entre le simulateur et un bot.

- Simulateur : exécute la consommation d'énergie pour un BOT
- BOT : détermine le niveau de scan du terrain de 0 à 255
- Simulateur : le scan à lieu
- BOT : Détermine une action (aucune, déplacer, tirer, bouclier, se cacher, …)
- Simulateur : exécution de l'action et applique les changements s'il y en a

Le simulateur passe au BOT suivant.

Chaque BOT est prévenu immédiatement dès qu'une de ses caractéristiques est modifiée.

VIE

- Un BOT n’a qu’un seule vie.
- Il perd sa vie si son niveau d’énergie tombe à zéro. Il est alors purement et simplement déconnecté du serveur.

ENERGIE

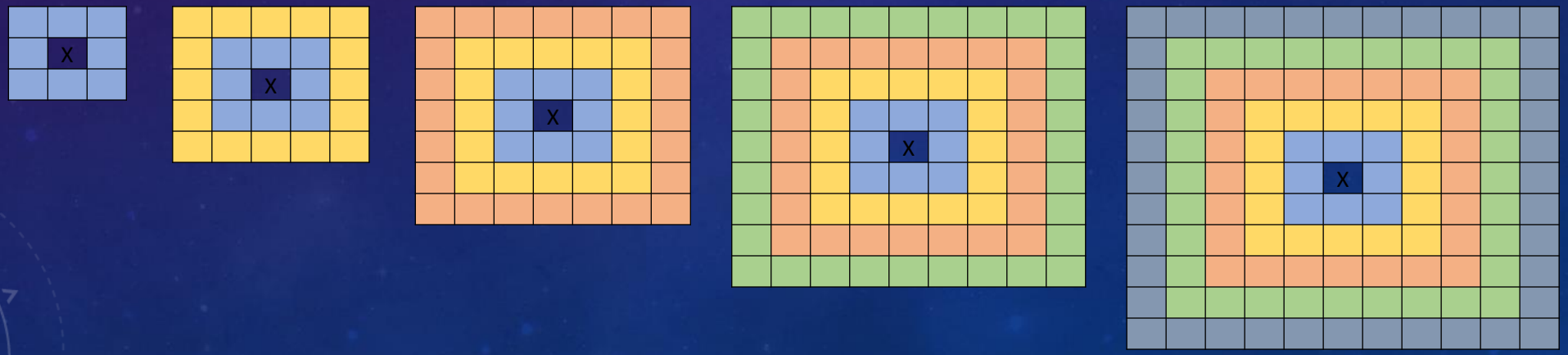
- Chaque BOT démarre avec 100 unités d’énergie
- Un BOT a perdu si sa réserve d’énergie tombe à zéro
- Il n’y a pas de limite de stockage de l’énergie
- Il y a perte au minimum de 1 unité d’énergie à chaque tour, même sans aucune détection, ni action effectuée
- L’énergie est consommé par la détection
- L’énergie est consommé par les actions, les chocs, les coups reçus
- De l’énergie peut être récupérée sur le terrain

CAPSULE D'ÉNERGIE

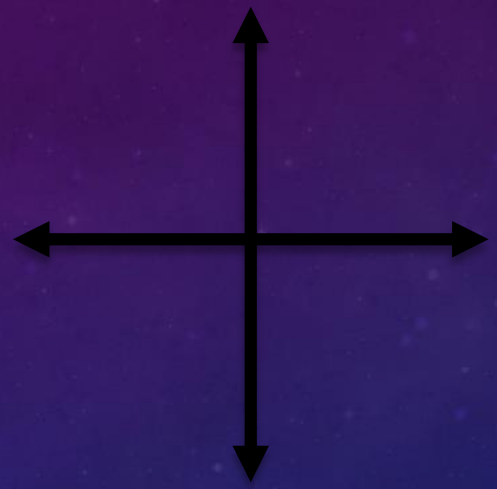
- Sur le terrain de jeu, il est possible de trouver des capsules d'énergie.
- Il suffit de passer sur une capsule d'énergie pour en récupérer le contenu.
- Des capsules d'énergie apparaitront de façon aléatoire sur le terrain.

SCAN DU TERRAIN

- Au début de chaque tour, il faut indiquer l'énergie injectée dans le capteur environnant
  - 0 : aucune information ne sera captée
  - 1 à 255 : les informations d'une distance de # case(s) seront captées. Cela correspond également à l'énergie consommée par ce scan !



DÉPLACEMENT



- Il est possible de se déplacer d'une case dans les 4 directions
- Il faut que la case soit vide ou contient une capsule d'énergie
- Si la case contient une capsule d'énergie : l'énergie contenu dans cette capsule est alors absorbée
- Si la case est occupé par un bloc : perte de 1 en énergie
- Si la case est occupé par un autre Bot, chaque Bot perd 1 en énergie

PROTECTION

- Il est possible d'activer un bouclier protecteur. On lui attribuera un certain niveau de puissance qui sera déduit en unité d'énergie pour l'activation
- A chaque tour il consommera 1 unité d'énergie pour maintenir son niveau de puissance
- Si l'on se prend 1 coup, il perd 1 unité de puissance
- Si sa puissance tombe à zéro, il est alors inactif
- Il est possible de réinjecter des unités d'énergie afin d'augmenter le niveau de puissance
- Lors de la désactivation du bouclier, on récupère en unité d'énergie le niveau de puissance actuelle du bouclier

CHAMP OCCULTANT

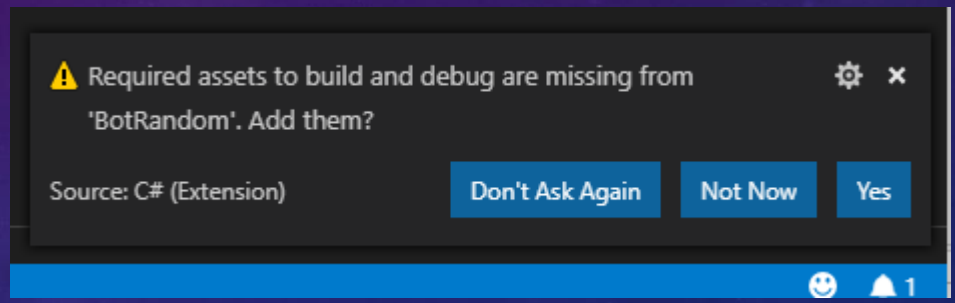
- Il est possible de se rendre invisible en activant un champ occultant. On lui attribuera un certain niveau de puissance qui sera déduit en unité d'énergie pour l'activation. Ce niveau de puissance est la distance où l'occultation est active.
  - Exemple : Avec un niveau 3, un ennemi à une distance <= 3 ne nous détectera pas, un ennemi plus loin que 3 pourra nous détecter
- A chaque tour il consommera 1 unité d'énergie pour maintenir son niveau de puissance
- Si l'on se prend 1 coup, il est automatiquement désactivé (et son énergie est perdue)
- Lors de la désactivation du champ occultant, on récupère en unité d'énergie le niveau de puissance du champ occultant

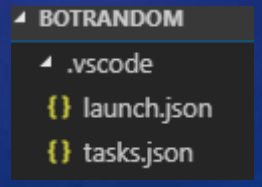
ENVIRONNEMENT DE DÉVELOPPEMENT

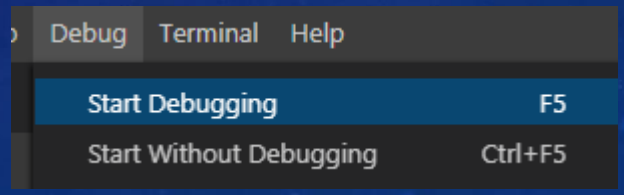
- Je conseille Visual Studio Code (multi-plateforme Windows, Linux, Mac)
  - Installation : https://code.visualstudio.com/docs/languages/dotnet
- Vous pouvez utiliser Visual Studio également

VISUAL STUDIO CODE

- mkdir BotRandom
- cd BotRandom
- dotnet new console
- code .
- accepter "tout"
- passer la commande
- dotnet dev-certs https --trust







EXEMPLE VIDE

- Voici le squelette d'un BOT en C#
  Il passe son tour à chaque fois.
- Un exemple un peu plus complet vous est également donné : le BOT essai de se protéger avec un bouclier et se déplace aléatoirement…
- En C# je fournis toute la gestion du protocole et la connexion au serveur.

```
public class TheBot
{

    // ***********************************************************************************************************
    // S'exécute une seule fois au tout début, avant le démarrage du jeu
    // C'est ici qu'il faut initialiser le bot
    public void DoInit()
    {

    } // DoInit

    // ***********************************************************************************************************
    /// Réception de la mise à jour des informations du bot
    public void StatusReport(UInt16 turn, UInt16 energy, UInt16 shieldLevel, UInt16 cloakLevel)
    {

    } // StatusReport

    // ***********************************************************************************************************
    /// On nous demande la distance de scan que l'on veut effectuer
    public byte GetScanSurface()
    {
        return 0;
    } // GetScanSurface

    // ***********************************************************************************************************
    /// Résultat du scan
    public void AreaInformation(byte distance, byte[] informations)
    {

    } // AreaInformation

    // ***********************************************************************************************************
    /// On doit effectuer une action
    public byte[] GetAction()
    {
        return BotHelper.ActionNone();
    } // GetAction


} // TheBot
```

PROTOCOLE POUR LE BOT

- Se connecter en WebSocket sur le serveur
- Envoyer un identifiant unique représentant le BOT (un guid)
- Le serveur répond « OK »
- Envoyer la trame « Nxxxxxxx » pour indiquer le nom « xxxxx » du BOT
- Le serveur envoi « T » : c'est un nouveau tour
  - il faut envoyer « D# » : indique que l'on veut une détection d'une surface de « # » (0 à 255)
- Le serveur envoi « I#xxxxx….xxxxx » : c'est le résultat de la détection
    - « # » est la surface de détection
    - « xxxxx….xxxxx » : tableau à 2 dimensions avec le détail de la détection
  - Il faut envoyer l'action que le BOT souhaite exécuter (voir exemple dans le code)
- Le serveur envoi « Ceessccpp » : les informations du BOT ont changées
  - ee : énergie / ss : niveau du bouclier / cc : niveau d'invisibilité / pp : score
- Le serveur envoi « D » : le BOT est détruit

```
// Si on ne veut rien faire, passer son tour
// return BotHelper.ActionNone();

// Déplacement du bot au nord
// return BotHelper.ActionMove(MoveDirection.North);

// Activation d'un bouclier de protection de niveau 10 (peut e
// return BotHelper.ActionShield(10);

// Activation d'un voile d'invisibilité sur une surface de 15
// return BotHelper.ActionCloak(15);

// Tir dans la direction sud
// return BotHelper.ActionShoot(MoveDirection.South);
```

PROTOCOLE POUR LE COCKPIT

- Se connecter en WebSocket sur le serveur
- Envoyer un identifiant unique représentant le BOT (un guid) à suivre
- Le serveur répond « OK »
- Le serveur envoi « Mwh################ » : il s'agit des informations du terrain de jeu
  - w : width, h : height
  - Le reste est un tableau à 2 dimensions indiquant si la case est vide (0) ou contient un mur
- Le serveur envoi « Nxxxxxxx » pour indiquer le nom « xxxxx » du BOT
- Le serveur envoi « Pxy » pour indiquer la position du BOT sur le terrain de jeu
- Le serveur envoi « I#xxxxx….xxxxx » : c'est le résultat de la détection
    - « # » est la surface de détection
    - « xxxxx….xxxxx » : tableau à 2 dimensions avec le détail de la détection
- Le serveur envoi « Ceessccpp » : les informations du BOT ont changées
  - ee : énergie / ss : niveau du bouclier / cc : niveau d'invisibilité / pp : score
- Le serveur envoi « D » : le BOT est détruit

LE COCKPIT

- L’idée de base est de représenter le tableau de bord de votre BOT
- Je ne donne aucun détail en plus du protocole pour laisser libre cours à votre créativité
- Les outils à votre disposition
  - Html
  - Css
  - Javascript

LE SERVEUR EST CONFIGURABLE

- Fichier settings.json
  - Le port pour WebSocket
  - La dimension du terrain de jeu
  - Le pourcentage de remplissage avec des blocs
  - Le pourcentage de remplissage avec des capsules d'énergie
  - Le coût en énergie de chaque événement
  - Le gain de points

- Cela permet d'affiner le gameplay…
  C'est ok pour des changements, si tout le monde les appliquent !


```
{
  "ServerPort": 4226,
  "MapWidth": 32,
  "MapHeight": 22,
  "MapPercentWall": 3,
  "MapPercentEnergy": 5,
  "EnergyStart": 100,
  "EnergyLostByTurn": 1,
  "EnergyLostByShield": 1,
  "EnergyLostByCloak": 1,
  "EnergyLostByMove": 1,
  "EnergyLostShot": 2,
  "EnergyLostContactWall": 5,
  "EnergyLostContactEnemy": 15,
  "PointByTurn": 1,
  "PointByEnergyFound": 8,
  "PointByEnnemyTouch": 20,
  "PointByEnnemyKill":  70
}
```

- Organiser des tournois inter-équipes
  - Avoir un classement des BOTs (meilleurs score, le highlander, …)

- Soutenance : Présentation
  - de votre organisation
  - de la logique de l’I.A. pour chacun des BOTs développés
  - du / des cockpits réalisés