



---

## Projet de fin d'études

*Bot Among Us*

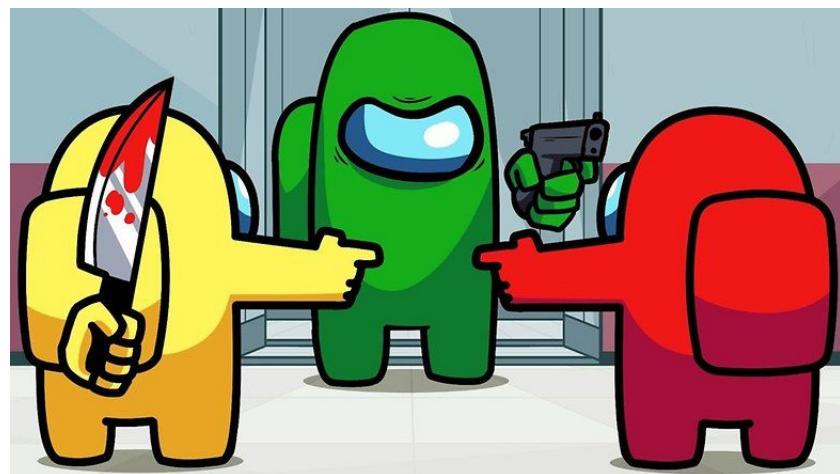
---

*Auteurs :*

Arnaud DETCHENIQUE  
Florent DENAT

Julien GARCIA

Sylvain IRIBARNE



## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Le jeu : règles et environnement</b>	<b>2</b>
2.1	But du jeu . . . . .	2
2.2	Règles du jeu . . . . .	2
2.3	Exploitation du jeu . . . . .	3
<b>3</b>	<b>Fonctionnalités du bot</b>	<b>4</b>
3.1	Architecture du bot, analogie avec le corps humain . . . . .	4
3.2	Déplacement dans l'environnement . . . . .	5
3.2.1	Résolution technique - Optimisation multi-objectif de chemin . . . . .	5
3.3	Résolution des tâches . . . . .	7
3.3.1	Résolution technique - Système expert . . . . .	7
3.4	Lecture du texte . . . . .	8
3.4.1	Résolution technique - OCR . . . . .	10
3.5	Reconnaissance des autres joueurs . . . . .	10
3.5.1	Résolution technique - <i>Object detection and localization</i> . . . . .	10
3.6	Mémorisation des évènements . . . . .	12
3.6.1	Résolution technique - Journal d'évènements . . . . .	12
3.7	Résumé vocal des évènements vécus . . . . .	13
3.7.1	Vocalisation . . . . .	13
3.7.2	Génération de phrases . . . . .	13
3.7.3	Résumé des évènements . . . . .	14
3.7.4	Résultats et optimisations . . . . .	15
<b>4</b>	<b>Conclusion</b>	<b>16</b>
4.1	Résumé . . . . .	16
4.2	Fonctionnalités à ajouter . . . . .	17
4.3	Mise en perspective . . . . .	18
<b>5</b>	<b>Bibliographie</b>	<b>18</b>

# 1 Introduction

La création d'un *bot* (abréviation de robot) dans un jeu vidéo peut servir diverses causes. En effet, on peut s'en servir pour faire des expériences et des études sur un environnement bien délimité, avec des règles précises et des conséquences identifiables. Un agent évoluant dans le jeu *Among Us* évoluera ainsi dans un monde social, confronté à des risques (la mort) et des récompenses (accomplissement des tâches et survie). Dans un premier temps l'agent ne servira qu'à simuler un joueur, dans le futur il pourra être utilisé afin de l'entraîner à résoudre des problèmes au sein du jeu, afin d'étudier les comportements humains lors des phases de débat ou bien afin d'étudier des mécanismes d'apprentissage par renforcement.

Dans un premier temps nous détaillerons les règles du jeu et de l'environnement, puis nous détaillerons chaque fonctionnalité implémentée ainsi que l'aspect technique associé.

## 2 Le jeu : règles et environnement

### 2.1 But du jeu

Le jeu se déroule dans un espace clos, divisé en pièces et couloirs. Un équipage s'y trouve et parmi eux on trouve un ou plusieurs imposteurs. Leur but est de tuer tous les membres de l'équipage. Les *crewmates* doivent réaliser toutes leurs tâches et/ou démasquer les imposteurs. Les imposteurs n'ont pas de tâches, ils doivent simuler le fait qu'ils sont en train d'en réaliser une pour se faire passer pour un crewmate. Ils peuvent aussi saboter le vaisseau pour accroître la difficulté des crewmates à atteindre leur objectif.

Lorsque quelqu'un veut signaler un cadavre ou appuie sur le bouton d'urgence, une phase de vote se lance. Durant celle-ci, l'équipage peut discuter et voter pour celui qu'ils veulent éjecter du vaisseau. Celui avec le plus grand nombre de voix est expulsé, et meurt (mais ne perd pas si c'est son camp qui remporte la partie).

### 2.2 Règles du jeu

La première règle du jeu et la plus importante est qu'il ne faut pas parler hors phase de vote. La seconde règle, plus évidente, est qu'il ne faut pas dévoiler son rôle d'imposteur.

Les imposteurs disposent d'un *cooldown* (temps d'attente) de recharge pour tuer et saboter. Le bouton 'tuer' apparaît lorsque ce cooldown est à 0 et qu'un crewmate est proche. Tout le monde peut, s'il trouve un cadavre, le signaler à l'aide du bouton 'Report' qui apparaît. Une phase de vote commencera alors.

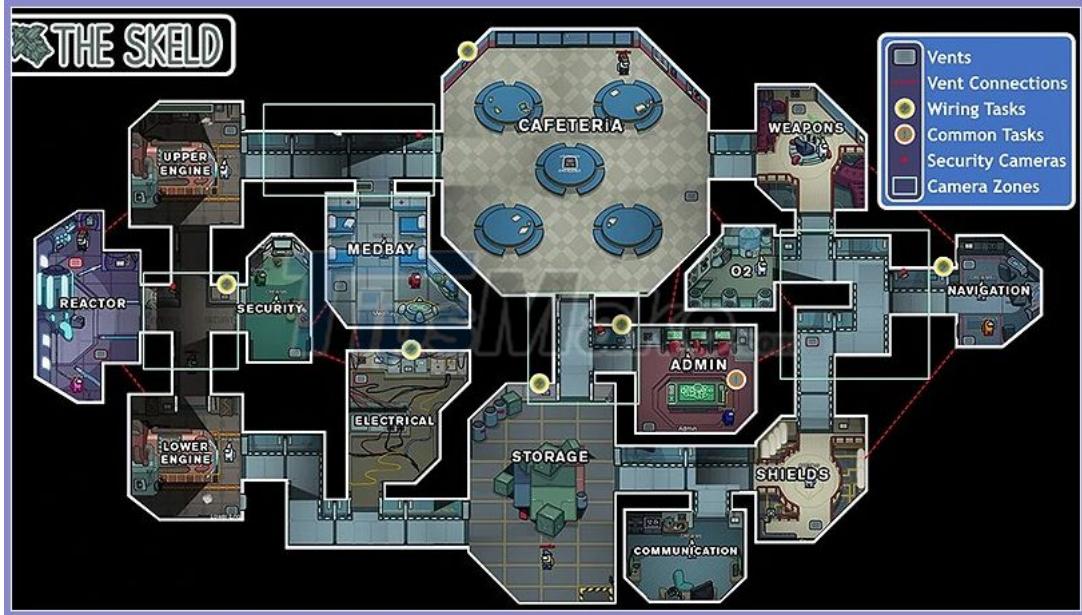


FIGURE 1: Carte de jeu



FIGURE 2: Phase de vote

Certaines tâches ne sont pas disponibles dès le départ, par exemple les sabotages qui sont créés de façon malicieuse par les imposteurs, ou encore les tâches qui se font en plusieurs étapes.

## 2.3 Exploitation du jeu

Deux possibilités pour la création de l'agent sont à identifier.

- La création d'un *mod* du jeu (accès aux ressources du jeu)
- L'interfaçage avec le jeu comme un humain (accès à l'écran, au clavier et à la souris)

La création d'un *mod* permet d'avoir accès aux ressources du jeu. Ainsi, toute la logique n'est pas à recoder. On peut par exemple se servir directement des attributs *room*, ou *kill\_cooldown* pour connaître précisément ces informations. Cependant, cela implique de lier fortement le *bot* avec une version fixe du jeu, et enlève les possibilités de jouer en ligne.

La deuxième option permet d'abstraire le bot pour qu'il joue réellement comme un humain, avec les mêmes informations. Tout doit être déduit depuis la vision de l'écran (avec des screenshots) et les actions ne peuvent être exécutées qu'avec le clavier et la souris. Pour obtenir la *room* et le *kill\_cooldown* il faut donc les déduire de l'écran (en connaissant la position de l'agent ou en lisant le texte sur l'écran par exemple).

## 3 Fonctionnalités du bot

### 3.1 Architecture du bot, analogie avec le corps humain

L'agent doit être capable d'évoluer au sein de l'environnement et d'effectuer des tâches en parallèle. Comme un humain, il peut marcher tout en regardant devant lui et en faisant des calculs. Nous avons choisi une architecture en micro-services afin de rendre les différents services indépendants. Par exemple, le service *muscle* peut faire avancer l'agent pendant que le service *vision* scanne l'écran pour détecter d'autres joueurs.

Les différents services développés sont les suivants :

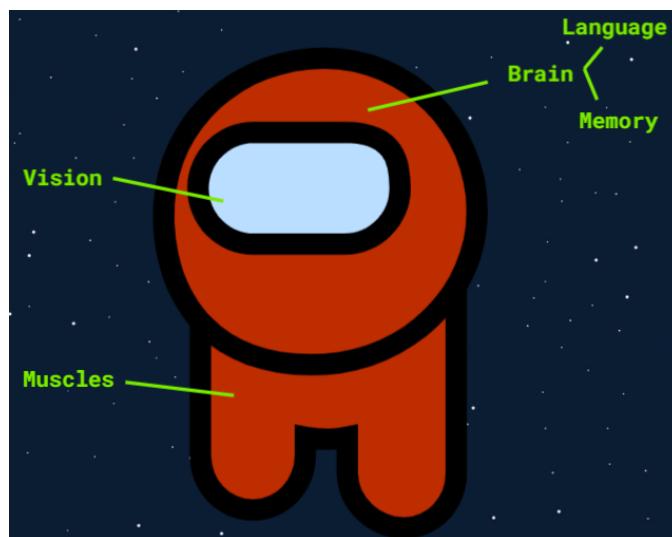


FIGURE 3: Les différents micro-services développés

- *muscle* : Déplace l'agent, exécute les tâches (envoie des commandes clavier et des clics souris)

- *vision* : Analyse l'écran en continu, détecte les changements de bouton, de phases de jeu, et détecte les joueurs qui apparaissent et disparaissent (envoie des évènements au *brain*).
- *brain* : Reçoit les évènements venant de ses sens (*vision, hearing*) et réagit en conséquence. Envoie des ordres aux muscles pour marcher, faire des tâches ou parler.
- *memory* : Stocke les évènements vécus par l'agent (voir la structure des logs en 3.6.1).
- *language* : Synthétise des phrases à partir d'information, des logs en l'occurrence. Résume les évènements vécus pour en faire un texte concis.

## 3.2 Déplacement dans l'environnement

L'agent se déplace d'un point A à un point B sur la carte. Le point A étant sa position actuelle et le point B peut représenter les coordonnées d'une tâche ou d'un objet à atteindre (autre joueur par exemple). La carte étant grande, il peut mettre un petit moment pour trouver le meilleur chemin à emprunter. Par analogie avec la partie précédente, il s'agit de ses muscles.

### 3.2.1 Résolution technique - Optimisation multi-objectif de chemin

Pour se déplacer d'un point A à un point B, l'agent ouvre la carte, via la touche 'tab', puis trouve les coordonnées de son logo sur l'image de la carte.



FIGURE 4: Logo de l'agent sur la carte



FIGURE 5: Carte ouverte en cours de partie

Une fois qu'il connaît sa position il doit trouver un chemin pour s'y rendre. Pour cela nous avons dû redessiner la carte pour indiquer quel chemin est empruntable. Nous avons donc le code couleur suivant :

- Noir : inaccessible (mur)
- Rouge : empruntable
- Blanc : empruntable et idéal. Permet à l'agent de ne pas trop frôler les murs, pour gagner en robustesse.



FIGURE 6: Décor en vue dézoomée

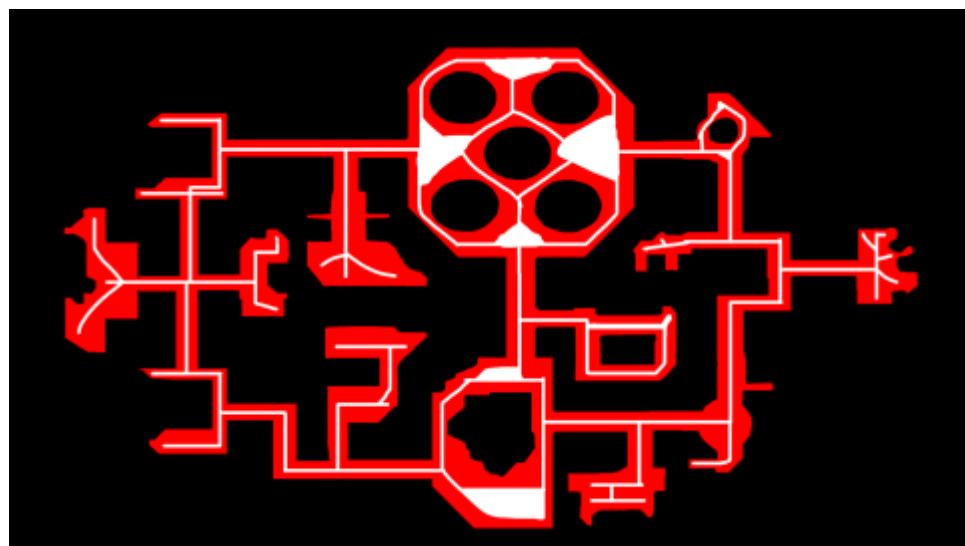


FIGURE 7: Carte redessinée

**Calcul du chemin** Chaque couleur a son propre poids afin de pouvoir utiliser l'algorithme A\* qui est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final. La version multi-objectif que nous avons utilisé permet d'accepter un compromis entre la distance totale du chemin et la concordance avec les zones blanches. Nous avons également ajouter une contrainte de *pruning* à A\* pour qu'il retourne un chemin rapidement même si ce n'est pas l'optimum. En effet, nous pouvons accepter de prendre un chemin plus long tant que l'on arrive à destination.

**Transformation du chemin en déplacement** Une fois le chemin trouvé nous transformons les distances en temps et lui indiquons comment se déplacer (ex : 3 secondes vers le haut, 2 secondes vers la gauche, etc).

Concernant le déplacement vers les tâches, nous avons renseigné dans une énumération toutes les informations des tâches et leurs coordonnées. Pour le déplacement vers les éléments du décor ou les autres joueurs à l'écran on calcule les coordonnées de la cible grâce à la distance visible entre les éléments (on connaît les coordonnées de l'agent).

**Correction de l'erreur de déplacement** Nous réouvrons régulièrement la carte pour vérifier si l'agent est réellement là où il pense être. Nous pouvons ainsi détecter les erreurs de déplacement pour les corriger, même si elles sont de l'ordre de 0,1%.

### 3.3 Résolution des tâches

Une fois arrivé au lieu de la tâche, il est capable de la résoudre. La première action consiste à cliquer sur le bouton *Use* qui apparaît lorsque le crewmate s'approche d'une tâche qu'il peut réaliser. C'est encore ses muscles qui sont en action.

#### 3.3.1 Résolution technique - Système expert

Si on en compte une cinquantaine avec les sabotages, quelques unes se répètent. Pour chacune, nous avons identifié un comportement de résolution et l'avons encodé. Il s'agit donc d'un système expert.

Essentiellement on distingue 3 types de tâches :

- Cible fixe : celles où la seule chose à faire est de déplacer la souris à des endroits fixes.  
Comme par exemple le scan de la carte dans la salle 'Admin'

- Cible à couleur fixe : celles où il faut détecter des couleurs puis effectuer une action, c'est le cas des fils électriques ou de la réparation du bouclier par exemple
- Texte à lire : celles où on doit lire une information à l'écran puis agir



FIGURE 8: Exemple de tâche avec couleur fixe : Nettoyage du filtre O2

Certaines tâches sont plus complexes encore. Par exemple, le sabotage des communications, pour lequel un petit algorithme basé sur de la trigonométrie permet de réussir cette tâche au moins 3 fois plus vite qu'un humain. On peut aussi citer la tâche 'Détruire astéroïdes' où l'on a jugé que la détection d'objets était inadéquat. Nous avons opté pour un spam de clic de souris, plus efficace.

### 3.4 Lecture du texte

**Lecture pour la résolution d'une tâche** Pour en revenir aux tâches où il faut lire du texte, nous avons utilisé l'outil *tesseract*. Par exemple, pour remettre l'oxygène il est nécessaire de lire un code numérique, pour débloquer le réacteur il est nécessaire d'ordonner une suite de nombres à l'écran. Au final l'agent est capable d'obtenir n'importe quelle information écrite à l'écran grâce à ce sens de vision.

**Lecture pendant l'évolution dans l'environnement** Pendant que l'agent se déplace dans son environnement des événements extérieurs peuvent survenir. En effet, les imposteurs peuvent saboter le vaisseau. De ce fait, à chaque instant il est nécessaire de vérifier qu'aucun incident n'est en



FIGURE 9: Exemple de tâche avec lecture de texte : Tâche du sabotage O2

cours. Pour cela, il est nécessaire de lire l'onglet des tâches pour identifier si un sabotage est en cours, et le cas échéant identifier lequel afin de s'y rendre.

Afin de ne pas surcharger les processeurs quand l'agent évolue nous avons choisi de délocaliser les calculs, pour cela nous avons utilisé l'API Google Cloud Vision. En effet, l'attente du résultat de la requête est une opération gratuite en terme de calcul.

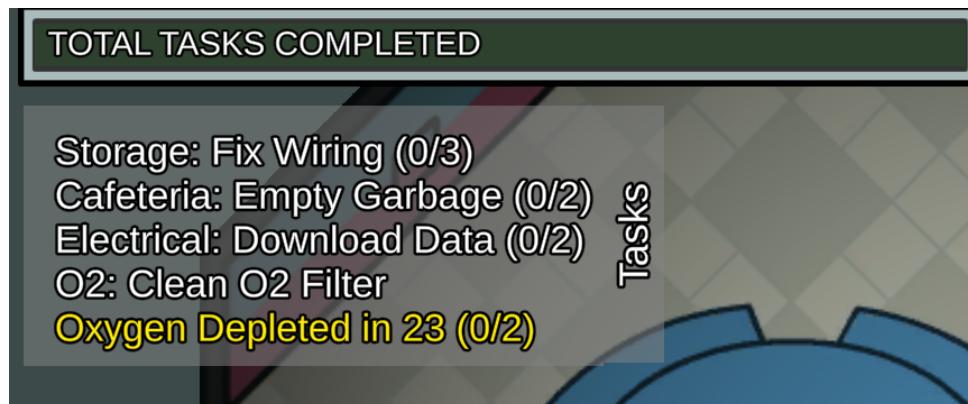


FIGURE 10: Liste des tâches comprenant un sabotage

### 3.4.1 Résolution technique - OCR

Pour lire un texte affiché à l'écran, la première étape consiste à délimiter les zones du texte. On applique un traitement d'image pour transformer une image en diagonale en image horizontale sans déformer les nombres inscrits.



FIGURE 11: Rotation d'image pré-OCR

Ensuite l'OCR, qui est déjà entraîné, renvoie les caractères inscrits sur l'image.

## 3.5 Reconnaissance des autres joueurs

En parallèle de ses déplacements et actions, le bot est capable de reconnaître l'environnement autour de lui. Il détecte quel joueur se trouve dans son champ de vision, et s'il y a un (ou plusieurs) cadavre(s). Il identifie également la couleur de chaque personnage.

### 3.5.1 Résolution technique - *Object detection and localization*

**Modèle de détection d'objet : RetinaNet** Le modèle [RetinaNet](#) apporte deux améliorations par rapport aux autres modèles de détection d'objets : Les *Feature Pyramid Networks* et le *Focal Loss*. Le papier ci-dessus démontre l'efficacité du modèle pour des datasets d'images satellites. De la même façon, les objets avec lesquels nous traitons sont de petite échelle et parfois très représentés à l'écran (10 joueurs maximum).

Nous avons fait du *Transfer Learning* sur un RetinaNet pré-entraîné sur le dataset COCO. Ainsi, avec notre dataset de 500 images annotées et un temps de fine-tuning d'une heure, nous obtenons une précision de 85%.

Il est important de noter que la prédiction avec ce modèle prend plus d'une seconde lorsque d'autres threads sont exécutés en même temps sur les machines sur lesquelles nous avons testé. En effet, nous avons privilégié la précision au détriment du temps de calcul. Nous préférions connaître de manière certaine les joueurs présents toutes les deux secondes plutôt que d'avoir des erreurs en prédisant en continu.

Afin d'utiliser ce modèle nous avons ajouté une brique de post-traitement. En effet, nous avons ajouté la correction d'erreur ainsi que la détection de la couleur des joueurs.

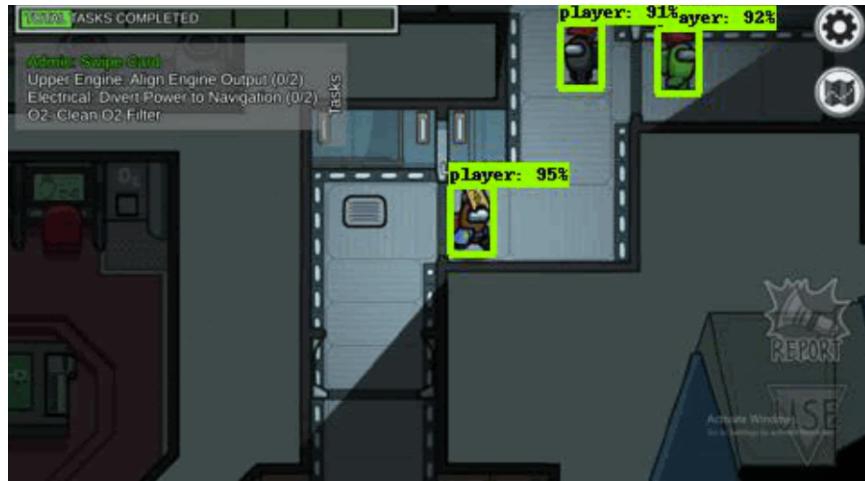


FIGURE 12: Détection des joueurs

**Post-traitement - Identification de la couleur** Pour reconnaître la couleur de chaque joueur que l’agent croise nous regardons la couleur dominante de chaque box. Le port des skins ne dérange pas cette méthode car les skins ont volontairement des couleurs différentes des couleurs de l’équipage pour limiter les confusions.

**Prise en compte de l’ombre sur les joueurs** Plus un membre de l’équipe est loin dans notre champ de vision plus il est dans l’ombre. Nous avons donc dû comprendre comment l’ombre modifiait les couleurs et ainsi procédé de la même façon afin de pouvoir détecter les couleurs des joueurs tapis dans l’ombre.

Nous avons donc analysé l’évolution de la couleur des pixels afin d’extraire la logique du filtre qui est appliqué. Nous avons déterminé que la variation des couleurs à une distance donnée était linéaire selon la relation suivante, valable pour les trois composantes de couleurs, pour n’importe quelle couleur de skin. Elle a été obtenue grâce à une régression linéaire.

$$0.51x + 7.3$$

On a ainsi pu annuler l’effet du filtre sur les couleurs.

**Détection des cadavres** La détection des cadavres se fait également en post-traitement. Pour cela, on recherche l’image de l’os à l’intérieur de la box.

**Correction d’erreur** Le post-traitement décrit ci-dessus permet d’identifier les box qui ne contiennent pas de joueurs. En effet, lorsqu’aucune couleur de joueur n’est représentée plus que les autres on déduit que la box n’est pas un joueur.



FIGURE 13: Equipier proche de l'agent - Pas d'ombre



FIGURE 14: Equipier loin de l'agent - Ombre très marquée

## 3.6 Mémorisation des évènements

L'agent va mémoriser toutes les actions qu'il "vit". Il va se souvenir des salles qu'il a traversé, des actions qu'il a réalisé, des personnes qui sont apparues dans son champ de vision ainsi que celles qui ont disparu.

### 3.6.1 Résolution technique - Journal d'évènements

Pour se souvenir des informations importantes durant la partie, nous avons utilisé un système de logs. On dénombre 3 types de logs :

- lors de la résolution de tâche (ou lorsqu'on fait semblant pour un imposteur)
- lorsqu'on détecte un joueur, vivant ou mort, en plus ou en moins par rapport à la dernière reconnaissance de joueurs
- lorsqu'on change de pièce

**Structure des logs** Un log contient donc le temps passé depuis le début du round, la salle dans laquelle il est, les personnes vivantes et mortes autour de lui et la tâche qu'il a pu réaliser.

room	time	players	killed	task
	102.07304525375366	[]	[]	Début de round
Weapons	10.961079597473145	[]	[]	
Weapons	25.021607637405396	[]	[]	Clear Asteroids
Shields	49.80301022529602	[]	[]	
Shields	56.552385568618774	[]	[]	Prime Shields
Storage	76.06380772590637	[]	[]	
Admin	89.20988917350769	[]	[]	
Storage	93.22875952720642	[]	[]	
Admin	102.23594617843628	[]	[]	
Admin	111.06562423706055	[BLUE_PLAYER]	[]	
Admin	113.96600008010864	[]	[]	Swipe Card
Admin	120.85446906089783	[]	[BLUE_PLAYER]	

FIGURE 15: Exemple de log généré

Nous créons les logs grâce à un système d'évènement. C'est à dire que chaque ligne est écrite quand il y a un changement. Ceci nous permet de ne pas avoir de répétition, ainsi chaque log apporte une information utile.

**Connaître la salle courante** Nous connaissons à chaque instant la position de l'agent, nous connaissons également les coordonnées de chaque pièce qui sont renseignées comme des objets Polygon (module python Shapely). Il nous est donc facile de vérifier dans quelle salle est l'agent et ainsi de déclencher l'évènement s'il y a un changement.

**Connaître les joueurs présents** Pour savoir si un joueur est visible (ou n'est plus visible), nous comparons le dernier log avec les joueurs que nous détectons grâce à notre modèle de détection de joueurs. S'il y a une différence nous la notons.

**Connaître la tâche effectuée** A chaque fois qu'une tâche est effectuée nous ajoutons l'entrée correspondante au journal d'évènements.

## 3.7 Résumé vocal des évènements vécus

Une fois que toutes les informations sont récoltées, le cerveau fait le tri. Il regarde les différences entre chaque ligne pour noter les évènements qui se sont déroulés autour de lui durant la phase de jeu. Par exemple, il peut noter que le joueur Bleu n'est plus à côté de lui ou qu'il a changé de pièce. Il les transforme ensuite en un texte concis qu'il va exprimer aux autres membres de l'équipage. Seules les informations utiles à la recherche des imposteurs sont transmises.

### 3.7.1 Vocalisation

Afin de faire oraliser l'agent nous avons ajouté une connexion à *Discord*, une plateforme d'échanges écrit et vocal. L'API Discord permet de créer un utilisateur robot pouvant écrire ou lire des .mp3. Nous avons donc utilisé le module *Text-To-Speech* de Google pour transformer les phrases générées (détailé ci-dessous) en fichier .mp3. Ces phrases sont ensuite lues par le robot sur Discord.

### 3.7.2 Génération de phrases

Passons maintenant à la retranscription des évènements de l'agent. Tout d'abord il pré-process ses logs, en éliminant les logs redondants ou inutiles, et en transformant ces logs en phrases. Afin de générer de la diversité et des phrases chaleureuses, nous avons choisi de créer une énumération de

phrases possibles pour chaque évènement, avec des tournures de phrases différentes. L'évènement "entrer dans Cafétaria" peut être traduit par "I was in Cafeteria", par "I walked around Cafeteria with my heart open to the unknown." ou encore "I walked on the floor of Cafeteria." par exemple. Une fois que tous les évènements sont transcrit, on obtient un long texte, comportant des phrases courtes et redondantes, qui contient toutes les informations sur la phase de jeu précédente.

### 3.7.3 Résumé des évènements

Il serait trop long et redondant de dire ces phrases telles quelles. En effet, l'agent doit partager le temps de parole avec les autres joueurs. Pour pallier ce problème, nous avons choisi d'utiliser un modèle basé sur les Transformers pour générer un résumé de ce texte.

Tout d'abord, nous souhaitons générer un résumé d'abstraction, c'est-à-dire un résumé qui ne se contente pas de choisir certaines phrases de texte original, mais qui génère ses propres phrases. L'objectif est de générer des phrases qui vont contenir un maximum d'informations du texte original. Nous avons donc besoin d'un modèle de text-to-text performant, qui comprend le sens du texte original. Pour cela, les modèles basés sur des Transformers, et notamment GPT et T5 semblent tout indiqués. Nous avons écarté les modèles du style BERT, qui retournent généralement un label de classe ou un échantillon du texte en entrée. Au vu de nos objectifs, et du caractère peu commun des textes que nous avons à résumer, nous avons choisi le modèle T5, qui est, par conception, très adapté au Transfer Learning.

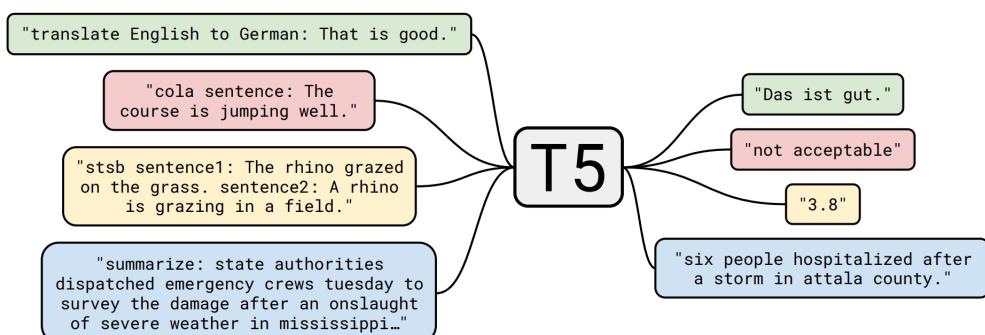


FIGURE 16: Modèle T5

Ce modèle a été entraîné une première fois sur un dataset extrêmement large du nom de C4. Afin de lui apprendre à faire des résumés, nous avons choisi d'effectuer un transfer learning sur 4515 articles déjà résumés sur Kaggle. Le modèle, une fois entraîné, est performant pour résumer des articles. Malheureusement, les phrases que nous avons générées précédemment sont assez différentes d'articles. Ce sont des phrases très courtes et redondantes, où l'information est sou-

vent contenue dans un seul mot (le nom de la pièce, de la tâche effectuée ou du joueur vu). Nous avons donc généré un dataset tiré de nos logs, que nous avons résumé manuellement. Grâce à ces données, nous avons ensuite pu continuer l'entraînement sur notre corpus.

### 3.7.4 Résultats et optimisations

Notre première méthode d'évaluation de la pertinence du modèle est purement subjective puisqu'elle consiste en une critique réalisée par nous même en comparant les textes générés et les textes originels. Nous considérons que notre modèle fait bien son travail, malgré quelques répétitions de phrases dans certains cas. Les phrases générées ont toujours du sens. On note qu'il n'y a pas d'erreurs morphologiques, syntaxiques ou sémantiques, mais certaines erreurs pragmatiques persistent.

Évidemment, nous voulions une méthode d'évaluation plus objective, c'est pourquoi nous avons choisi le ROUGE score. ROUGE est une métrique axée sur le recall qui calcule le chevauchement de n-grammes de la référence et des textes générés. Ce chevauchement de n-grammes signifie que le schéma d'évaluation est indépendant de la position des mots, hormis les associations de termes de n-grammes. Ici, nous utilisons n-rouge, le type de rouge le plus courant, ce qui signifie un chevauchement de n-grammes. Nous utilisons 2-rouge par défaut, qui compare les bigrammes. L'interprétation des résultats se fait de la manière suivante : Rouge-2 recall : x% signifie que x% des bigrammes dans le texte de référence sont aussi présents dans le texte généré. Rouge-2 précision : x% signifie que x% des bigrammes dans le texte généré sont aussi présents dans le texte de référence. Rouge-2 F1-score : C'est la moyenne harmonique de la précision et du recall. Il prend donc en compte les deux métriques précédentes et peut être calculé de la manière suivante :

$$\frac{2.0 * \text{precision} * \text{recall}}{\text{precision} + \text{recall} + 1e - 8}$$

Nos résultats sont calculés à partir des données de validation. Le texte de référence est un résumé créé par nos soins, et le texte généré a été créé de toute pièce par notre modèle.

Rouge-2 recall	0.595...
Rouge-2 precision	0.714...
Rouge-2 f1-score	0.649...

Nous avons des résultats plutôt corrects avec une précision et un f1-score supérieurs à 60%.

Voici un exemple de phrases résumées à l'aide de notre modèle.

```
I walked the floor of storage. I have walked the floor of cafeteria.
I entered storage. I entered electrical. I entered storage. I saw
Black. I saw more Black. I entered electrical. I have finished the
Calibrate Distributor task. I entered storage. I entered shields. I
entered oxygen. I entered weapons. I entered cafeteria. I entered
admin. I saw Blue. I saw more Blue. I finished the Swipe Card task.
I walked the floor of storage. I walked the floor of electrical. I
have finished the Calibrate Distributor task. I entered storage. I
have walked on shields soil. I entered oxygen. I have walked on
shields soil. I walked the floor of storage. I walked the floor of
electrical. I have finished the Calibrate Distributor task. I walked
the floor of storage. I entered shields. I entered oxygen. I entered
shields. I walked the floor of storage. I walked the floor of
electrical. I have finished the Calibrate Distributor task. I
finished the Divert Power task. I entered lower engine. I have
stepped on the floor of security. I saw Green. I saw more Green. I
finished the Accept Power (security) task.
```



```
I crossed Black in storage
and Blue in admin while
going to do Swipe Card
task. At the end I was
doing my tasks at lower
engine then electrical and
I crossed paths with Green.
Arnaud is suspect.
```

FIGURE 17: Exemple de résumé

Cet exemple est très satisfaisant. Malheureusement, il n'est pas toujours aussi bon et on peut retrouver parfois des redondances comme dans l'exemple suivant :

Texte résumé par nos soins : *I walked around upper engine with Blue, Lime and Orange, then I did my cafeteria and storage tasks. I crossed Orange to electrical, then Green to reactor. When I returned to electrical I saw Lime dead, it's Orange for sure. Arnaud is suspect.*

Texte généré : *I crossed Lime's backbone while walking around the upper engine, my heart open to the unknown. At the reactor, I ran into electrical and I crossed Lime's backbone. Then I crossed Lime's backbone. Arnaud is suspect.*

Concernant le temps de prédiction, un résumé de 150 mots nécessite 3 secondes de calcul à l'aide d'une GPU K80.

## 4 Conclusion

### 4.1 Résumé

Finalement, l'agent développé est autonome en tant que crewmate, il est capable de se déplacer, de finir ses tâches, et d'enregistrer ce qu'il voit. Il peut également signaler les cadavres qu'il croise.

En cas de sabotage il arrête ce qu'il fait pour aller réparer le vaisseau. Une fois qu'il a fini ses tâches, il patrouille dans le vaisseau afin de récolter des informations. Il communique ensuite toutes ces informations au reste de l'équipage.

Il n'est cependant pas encore capable de comprendre les arguments des autres joueurs afin de choisir pour qui voter.

En tant qu'imposteur il est capable de faire semblant d'être crewmate et de tuer des membres de l'équipage. Lorsqu'il croise un corps, il a 10% de chance de le signaler pour que personne ne sache s'il est imposteur ou non lorsqu'il "report" un cadavre. De la même manière qu'un crewmate, il patrouille dans le vaisseau afin de trouver des gens à tuer.

## 4.2 Fonctionnalités à ajouter

Ainsi, diverses fonctionnalités peuvent être ajoutées (cf. [le repository sur Github](#) pour la contribution au projet). Nous pouvons lister les suivantes par exemple :

- Accélérer le calcul du chemin : Il serait envisageable d'entraîner un modèle à prédire un chemin calculé en fonction de la carte et des coordonnées des points de départ et d'arrivée. Accélérer le calcul d'une tâche lourde grâce à un modèle d'IA est un sujet de recherche contemporain.
- Généraliser l'évolution dans l'environnement à n'importe quelle carte : En effet, nous avons défini nous-même la matrice de déplacement d'une carte du jeu. De plus, les positions des tâches ont été définies à l'avance.
- Résoudre de nouvelles tâches sans ajouter des règles au système expert : Cette fonctionnalité nécessiterait de rendre la résolution des tâches complètement généraliste. Il faudrait ainsi apprendre à l'agent comment résoudre les tâches selon des mécaniques plutôt qu'avec des positions hard-codées. Ceci serait envisageable en mettant en place un système d'apprentissage par renforcement. Maintenant que l'agent est autonome il est possible de lui faire faire un grand nombre d'essais pour évoluer au sein de l'environnement et obtenir des récompenses. Toutefois, la grande diversité des tâches et la complexité pour le récompenser rendent cette tâche ardue.
- Améliorer la vitesse de la détection des joueurs : Le modèle que nous avons choisi a une excellente précision mais présente le défaut de nécessiter du temps de calcul. Utiliser MobileNet pourrait accélérer les calculs en réduisant la précision. Maintenant qu'un système de correction d'erreur a été ajouté en post-traitement il est envisageable de choisir MobileNet.
- Ajouter des interactions avec les autres joueurs, une interface de question/réponse par exemple : Cette fonctionnalité de NLP permettrait à l'agent de communiquer plus efficacement sur des

événements précis. De plus, afin de simuler complètement un joueur l'agent doit pouvoir écouter les arguments des autres joueurs et en tenir compte.

### 4.3 Mise en perspective

Ainsi, ce projet aura permis de mettre en place un agent capable d'évoluer et d'être autonome au sein d'un environnement social dans un jeu de bluff. Il est donc maintenant possible d'étudier le comportement des joueurs dans cet environnement. A terme, l'agent pourrait être capable de déterminer quand les autres joueurs mentent ou disent la vérité. Il pourrait ainsi servir de coach pour les joueurs ou d'outil pour étudier les comportements de bluff, ou les réactions au stress au sein d'un groupe.

## 5 Bibliographie

1. Few Shot Object Detection for TensorFlow Lite.  
[https://github.com/tensorflow/models/blob/master/research/object\\_detection/colab\\_tutorials/eager\\_few\\_shot.ipynb](https://github.com/tensorflow/models/blob/master/research/object_detection/colab_tutorials/eager_few_shot.ipynb)
2. Retina U-Net: Embarrassingly Simple Exploitation of Segmentation Supervision for Medical Object Detection.  
<http://proceedings.mlr.press/v116/jaeger20a>
3. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.  
<https://arxiv.org/abs/1910.10683>
4. Fine Tuning Transformer for Summary Generation  
[https://github.com/abhimishra91/transformers-tutorials/blob/master/transformers\\_summarization\\_wandb.ipynb](https://github.com/abhimishra91/transformers-tutorials/blob/master/transformers_summarization_wandb.ipynb)
5. NEWS SUMMARY - Generating short length descriptions of news articles.  
<https://www.kaggle.com/sunnysai12345/news-summary>
6. ROUGE: A Package for Automatic Evaluation of Summaries  
<https://www.aclweb.org/anthology/W04-1013.pdf>