JOURDAN Luca n°11308293 - TOUATI Gaïs n°11307497 - FORT Tz'ayik n°11317334

Projet Dynamique des Connaissances

Trace Modeling and Visualization

I. Modélisation des traces

Dans le cadre de l'UE Dynamique des Connaissances, nous avons effectué une modélisation, analyse et visualisation de traces, décrivant les actions de plusieurs joueurs lors d'une partie de Minecraft. Les traces nous ont été fournies au format JSON.

La première étape a été de déterminer un modèle pour nos traces afin de les exploiter efficacement. Nous avons listé tous les types d'événement dans la trace fournie, que nous avons répartis en quatre catégories selon les similarités dans leur sémantique et dans leurs attributs (Figure 1). Par exemple, tous les obsels *BlockEvent* ont un attribut *m:blockName* en commun et les obsels *ItemEvent* ont un attribut *m:itemName* en commun.

Certains événements de la trace ont été légèrement modifiés pour coller à notre modèle. L'attribut *numberOfCraft* des événements de type *Craft* a ainsi été renommé *amount* pour que tous les obsels de type *ItemEvent* aient l'attribut *amount* en commun. Aussi, nous avons transvasé la valeur de l'attribut *time* des traces dans les attributs *begin* et *end* de notre modèle pour correspondre aux spécifications de ktbs. Nous avons aussi supprimé les attributs *data* et *dimension* pour leur inutilité dans notre analyse, et *@id* qui est automatiquement généré par ktbs lors du remplissage des traces. On a remarqué plus tard que l'axe z dans les traces correspond à la profondeur et non à la hauteur, cela aurait été intéressant d'inverser y et z directement dans la modélisation.

Une fois le modèle créé et la trace adaptée, nous les avons poussés sur le ktbs en tant que *model1* et *baseTraces* respectivement. Voici les liens vers le ktbs :

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/model1
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/baseTraces/

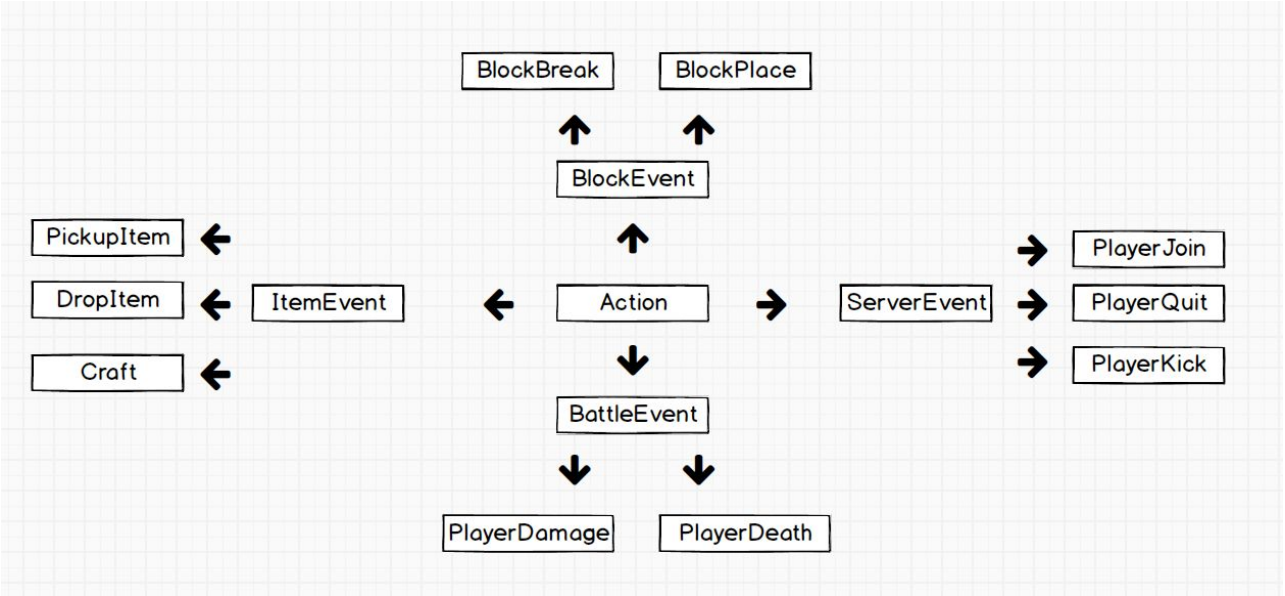

Figure 1- *Modèle simplifié des traces*

II. Analyse et transformation.

A) Les constructions

Minecraft étant un jeu de construction, la première question que l'on se pose naturellement est "Qu'ont construit les joueurs ?". Pour répondre à cette question nous avons effectué un premier filtre sur les obsels de types *BlockPlace* sur la trace *baseTraces* que l'on a nommé *filtered1*.
Resultat : https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/filtered1/

Nous avons ensuite crée un programme en Python pour visualiser le résultat. Ce programme effectue une requête HTTP GET sur les obsels de la nouvelle trace. On parcourt ensuite tous les obsels et on affiche leurs coordonnées dans un espace en 3D avec la librairie *pptk* sous la forme d'un point, dans une couleur dépendant du type de l'objet posé. (Figure 2)

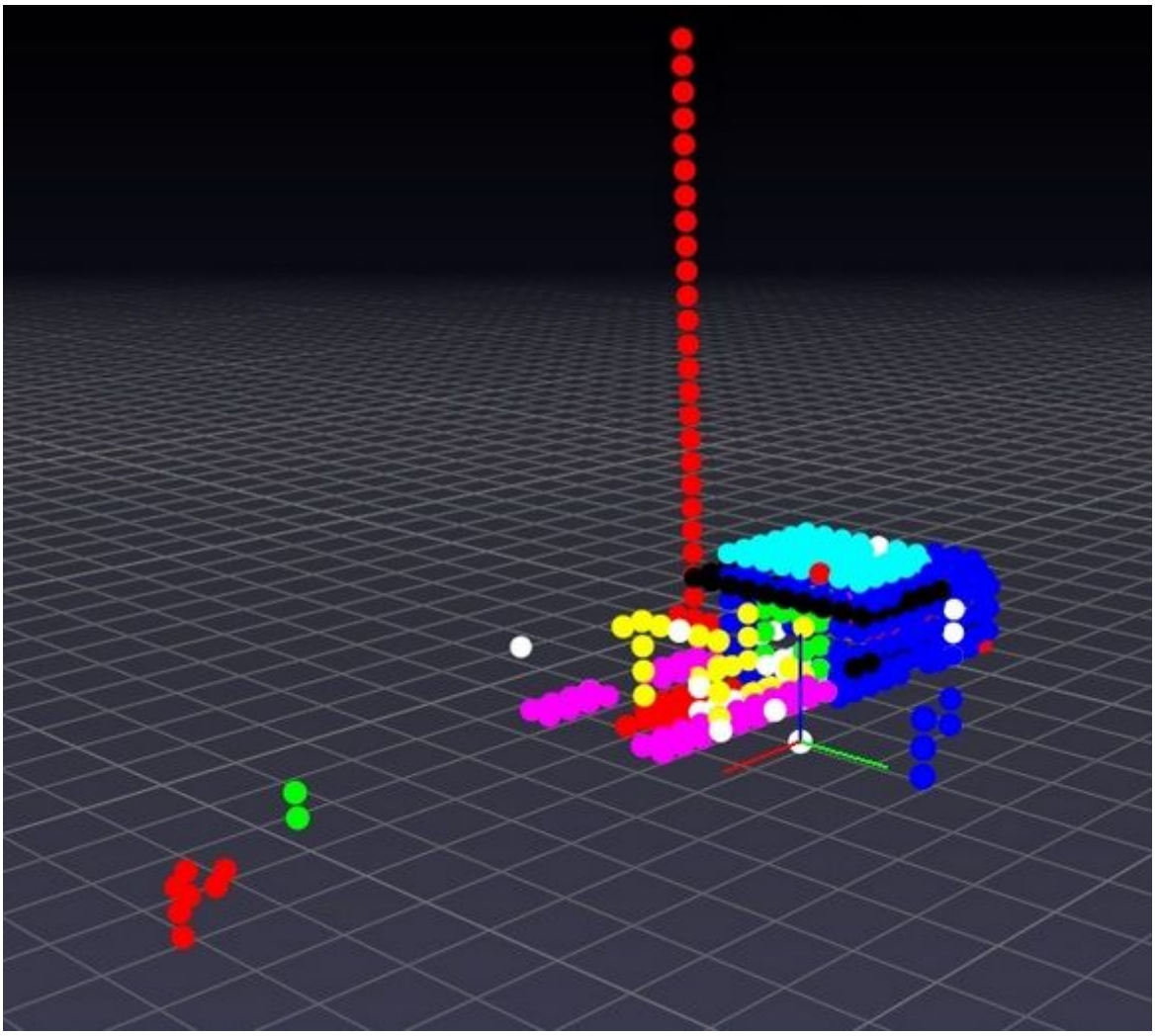

Figure 2 - *BlockPlace général*

On peut voir que les joueurs se sont construit un abri en bois (bleu foncé). Le toit de l'abris et un contour décoratif ont été fait en "escaliers" (bleu clair et noir) pour donner de la forme à leur maison. Devant, ils ont placé un parterre de fleurs (magenta) et une sorte d'arche en "spruce fence" (jaune). On observe que le terrain a été aplani avec quelques blocs de terre (rouge) avant de construire. La porte de la maison a été fait en bûches (vert) et ils ont construit une petite allée en argile (blanc) devant leur porte. On remarque aussi deux constructions étranges, une grande tour de terre (rouge) à côté de la maison, et au loin (non visible sur la figure 2), une autre tour de bois. D'autres éléments divers (blanc) ont été placés ponctuellement, comme un workbench qui a été déplacé plusieurs fois.

Nous nous somme ensuite demandé quelle était la participation de chaque joueur dans cette construction. Pour y répondre, nous avons créé un nouveau filtre pour chaque joueur sur la base du filtre précédent, pour observer les objets placés par chaque joueur. Résultat :

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/AliceBlockPlace/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/BobBlockPlace/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/CharlieBlockPlace/

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/DanBlockPlace/

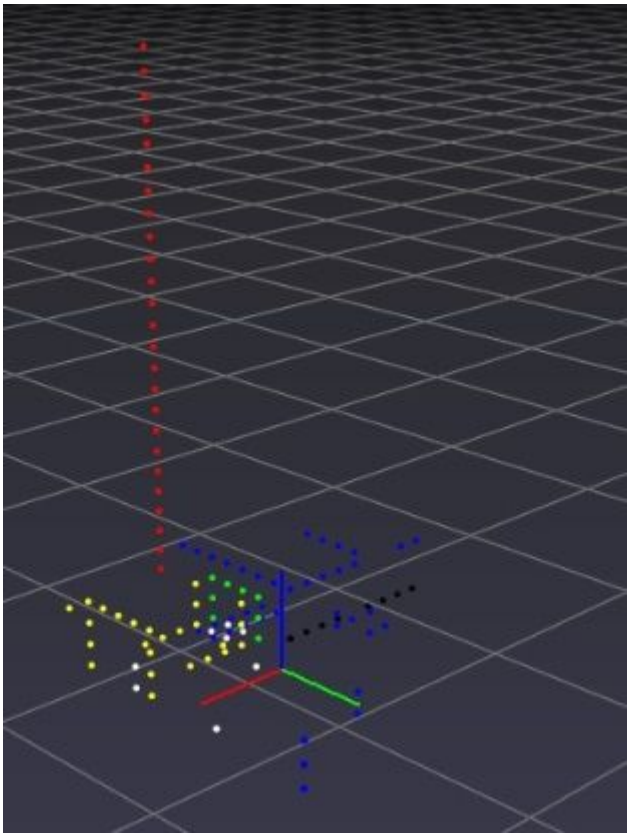

Figure 3 - *Exemple BlockPlace d'Alice*

On observe que Alice a construit une partie de la maison, l'arche, la porte et la tour de terre (Figure 3). Bob est le décorateur du groupe, il a posé le parterre de fleurs, et a fait le toit esthétique en escaliers. Charlie, lui, n'a pas participé du tout à la construction, mais on voit qu'il a construit une tour de bois au loin. Enfin Dan est le maçon du groupe, il a construit le plus gros du bâtiment.

Cette première approche a soulevé deux nouvelles question principales. Quelle est donc cette tour qu'Alice a construite ? Qu'a donc fait Charlie, perdu seul loin de ses camarades ?

B) Les blocs détruits

Pour répondre à la question sur Charlie, nous nous sommes intéressés aux blocs détruits. On supposait que Charlie s'était creusé un chemin sous terre, et la destruction des blocs pourrait nous le révéler. On en a aussi profité pour regarder l'activité de brisage de chaque joueur. Comme précédemment, nous avons fait un premier filtre sur la trace principale sur les obsels de type *BlockBreak,* puis une série de filtres pour chaque personnage.
Résultats :

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/BlockBreak/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/AliceBlockBreak/

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/BobBlockBreak/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/CharlieBlockBreak/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/DanBlockBreak/

Nous avons ensuite utilisé notre programme en Python pour visualiser le résultat (Figure 4).

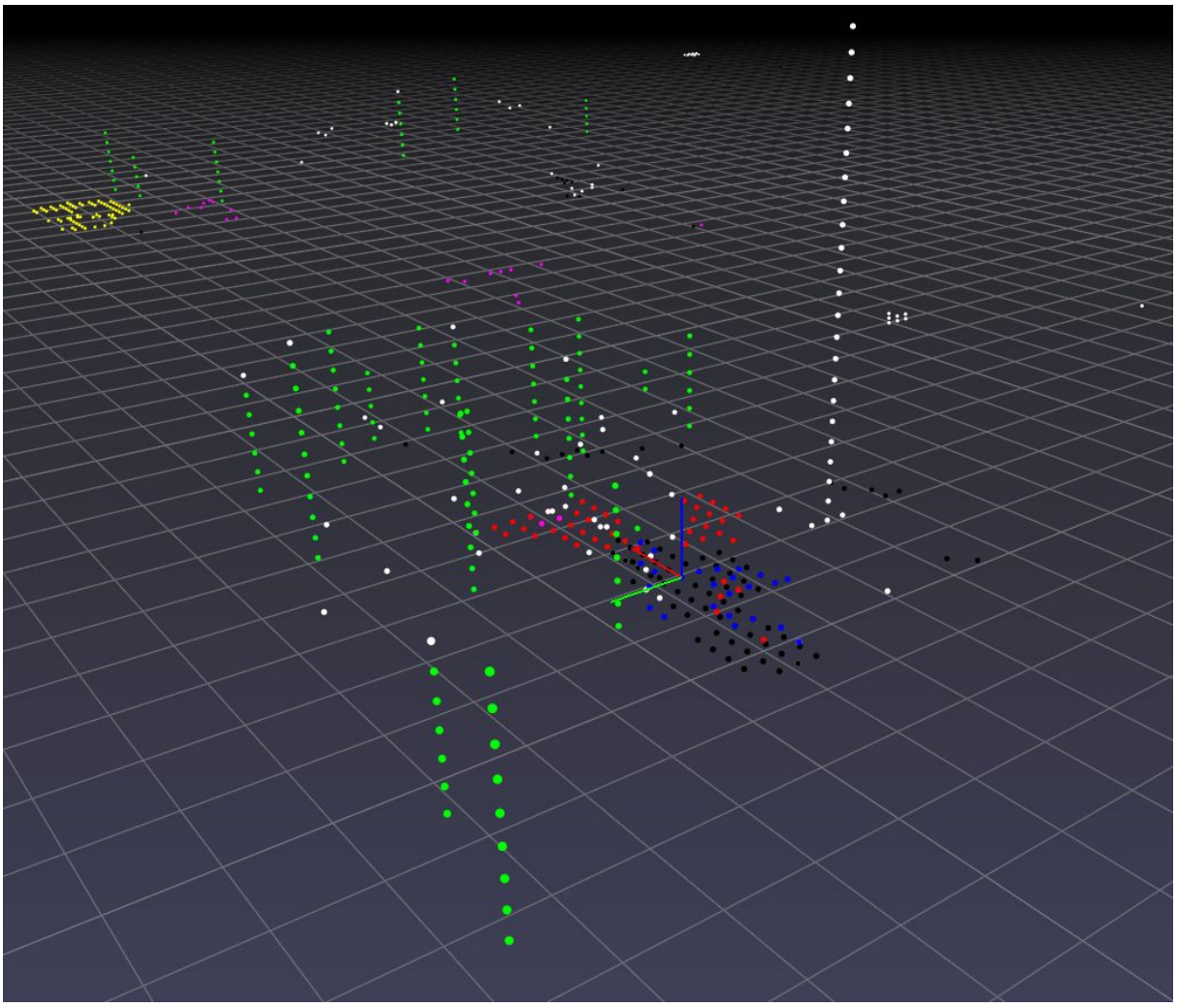

Figure 4 - *BlockBreak général*

On voit (au loin, non visible sur la figure 4), que Charlie n'a quasiment pas détruit de bloc non plus, bien qu'il ait réussi à descendre sous terre. On déduit que Charlie a dû trouver une grotte dans laquelle il s'est aventuré. Les événements dans cette grotte seront détaillés en partie C.

Ici, on peut voir que l'emplacement de leur maison a dû être défriché, des arbres ont dû être coupés (vert) et de l'herbe enlevée (noir). De plus, des blocs de neige (rouge) se trouvaient à l'emplacement de la maison ce qui nous indique qu'il s'agit d'un terrain neigeux. Nous pouvons également observer que la base de la maison, en bois (bleu), à été détruite par l'un des joueurs.

Nous voyons également qu'un joueur a ramassé du sable (jaune) et que des fleurs ont été cueillies (magenta). De manière plus étonnante nous pouvons voir que la tour construite par Alice à été détruite. Cela nous a étonné car l'on savait qu'Alice s'était suicidée du haut de cette tour (cf Partie C) et nous nous somme donc demandé comment elle avait pu accéder à nouveau au sommet de la tour pour la détruire.

Afin de répondre à cette nouvelle question nous avons décidé de visualiser l'évolution des destructions et constructions au cours du temps. Pour ce faire nous avons modifié notre script en python afin de pouvoir voir plus finement de quelle manière la partie s'est déroulée.

De cette visualisation nous avons pu observer que la tour d'Alice à été construite une première fois, puis Alice est tombée et a subi des dégâts de chute non létaux. Puis elle a décidé de remonter en haut de la tour. Pour ce faire, Alice a détruit la bas de la tour pour remonter sur le même axe, puis elle est monté progressivement en détruisant les blocs au dessus d'elle et en reconstruisant sous ses pieds. Elle a ainsi atteint de nouveaux sommets, puis elle chut à nouveau. Mais la hauteur de sa chute, cette fois, a causé sa mort ...

C) Les combats, les dégâts, et les morts.

On voit de manière évidente dans les traces que Alice est la seule à être morte, et qu'elle l'a été 2 fois. Nous avons donc décidé d'enquêter sur sa deuxième mort, et d'en profiter pour visualiser les dégâts subis par les joueurs en général.

Nous avons donc effectué de nouveaux filtres sur la trace principale, en suivant la même méthode que pour les deux parties précédentes, afin d'obtenir les obsels correspondant à des prises de dégâts.
Résultats :

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/AlicePlayerDamages/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/BobPlayerDamages/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/CharliePlayerDamages/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/DanPlayerDamages/

Nous avons ensuite utilisé notre programme en Python pour la visualisation.

Tout d'abord, les résultats confirment le suicide d'Alice. On voit que sa première mort a eu lieu au pied de sa tour, a la suite d'importants dommages de chute. Sa seconde mort est quant à elle le résultat d'un affrontement avec différentes créatures. Nous pouvons voir qu'elle est morte à la suite d'une *ENTITY_ATTACK (en vert sur la représentation)* après s'être battue avec plusieurs créatures, car elle a subi des dégâts de type *ENTITY_EXPLOSION (en*

jaune sur nos représentations) et de type *PROJECTILE (en magenta sur la représentation)* qui sont infligés par différentes espèces de monstres dans Minecraft.

Charlie, dans sa grotte, s'est lui aussi battu contre plusieurs créatures (Figure 5). Une représentation temporelle nous montre qu'après être bien descendu, Charlie a subi de nombreux dégâts de combats avec différentes entités. On observe qu'il est tombé dans la lave dans le feu de l'action ce qui lui a infligé des dégâts de type *FIRE_TICK (en rouge)*. On voit également que lors de sa descente dans la grotte il est tombé à plusieurs reprises (en cyan).

Les autres joueurs ont eu une partie plus tranquille, avec quelques affrontements et dégâts de noyade, mais rien de particulièrement remarquable.

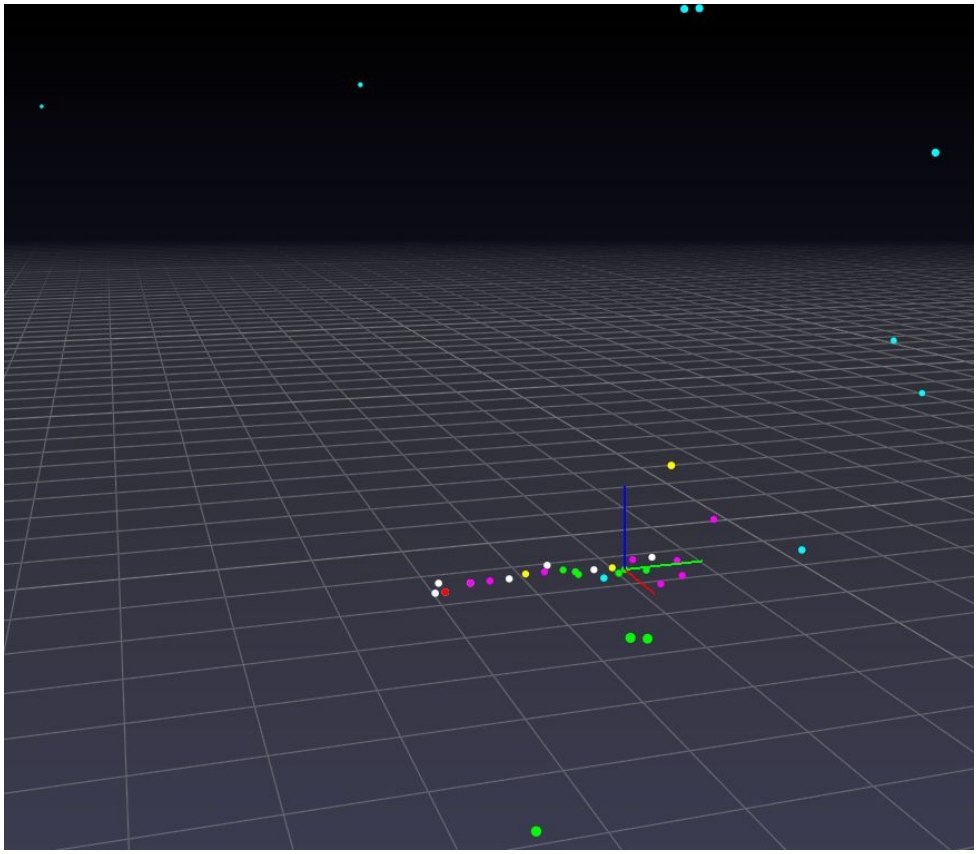

Figure 5 - *Exemple des dégâts de Charlie*

D) L'inventaire

Après avoir retracé le déroulement global de la partie, nous avons essayé d'effectuer un traitement plus complexe sur les traces pour obtenir plus de détails. Nous avons choisi de nous intéresser à l'état de l'inventaire des joueurs.

L'idée est la suivante : on liste tous les événements dans lesquels les joueurs peuvent obtenir de nouveaux items et tous les événements dans lesquels ils peuvent en perdre. Puis, en faisant la différence entre ces deux listes jusqu'à un instant t, on obtient le contenu de l'inventaire à cet instant : Objets gagnés - Objets perdus = Objets dans l'inventaire.

Premièrement, nous avons créé un nouveau modèle comprenant deux nouveaux types d'obsels : *NewItem* et *LoseItem* qui représentent les événements "gagner un item" et "perdre un item" et qui ont tous deux pour superType *ItemEvent*. Des nouveaux items peuvent être obtenus dans les événements *PickupItem* et *Craft*. Nous avons donc effectué deux réécritures pour créer des obsels de type *NewItem* à partir de ces deux types d'événement. Après plusieurs essais infructueux (qui justifient le nom donné à la première réécriture) avec les méthodes fsa et isparql, nous avons effectué ces réécritures avec des requête CONSTRUCT en sparql. (Figure 6)

```
CONSTRUCT
{
    [
        a m:NewItem;
        m:x ?x;
        m:amount ?amount;
        m:playerName ?playerName;
        m:y ?y;
        m:z ?z;
        m:itemName ?itemName;
        :hasTrace <%(__destination__)s>;
        :hasBegin ?begin ;
        :hasEnd ?end ;
        :hasSourceObsel ?o1.
    ] .
} WHERE {
  ?o1 :hasBegin ?begin.
  ?o1 :hasEnd ?end;
  m:x ?x;
  m:y ?y;
  m:z ?z;
  m:amount ?amount;
  m:itemName ?itemName;
  rdf:type m:PickupItem;
  m:playerName ?playerName
}
```

```
CONSTRUCT
{
    [ a m:NewItem;
      m:x ?x ;
      m:amount ?total ;
      m:playerName ?playerName ;
      m:y ?y ;
      m:z ?z ;
      m:itemName ?itemName ;
      :hasTrace <%(__destination__)s> ;
      :hasBegin ?begin ;
      :hasEnd ?end ;
      :hasSourceObsel ?o1
    ].
}

WHERE
{
   ?o1 :hasBegin ?begin ;
   :hasEnd ?end ;
   m:x ?x;
   m:y ?y ;
   m:z ?z ;
   m:resultType ?itemName ;
   rdf:type m:Craft ;
   m:playerName ?playerName ;
   m:numberOfCrafts ?nbcraft ;
   m:amount ?rbycraft.
   BIND(?numberOfCraft * ?resultAmountyCraft AS ?total).
}"
```

Figure 6 - *A gauche réécriture des PickupItem, a droite réécriture des Crafts*

On remarque qu'un petit calcul était nécessaire pour déterminer la quantité générée par un craft, qui vaut numberOfCraft * resultAmountByCraft.
Résultats :

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/samarchpas/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/NewItemCraft/

Puis pour obtenir la trace finale des NewItems, nous avons fusionné ces deux traces. Puis pour obtenir les NewItems pour chaque joueurs, nous effectuons à nouveau un filtre par joueur.

Résultat :

- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/NewItem/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/AliceNewItem/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/BobNewItem/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/CharlieNewItem/
- https://liris-ktbs01.insa-lyon.fr:8000/public/master-ia-2018/GTL/DanNewItem/

Pour finir, il aurait fallu faire le même raisonnement pour les LoseItems, mais c'est à ce moment que nous nous sommes rendu compte que notre modèle n'était pas correct. En effet, les ingrédients pour les crafts sont mal incorporés, et on ne peut donc pas déterminer quels items ont été perdus lors d'un craft. Nous avions modélisés les ingrédients comme un attribut de type xsd:list, ce qui n'est visiblement pas la bonne façon de faire. Nous pensons qu'une solution serait de créer un nouveau type d'obsels *Ingrédients* qu'on remplit pour chaque craft. Ces obsels contiendraient les ingrédients d'un craft, et seraient reliés à celui ci par une relation *recette*. Mais par manque de temps nous n'avons pas pu implémenter cette solution.

Nous avons tout de même profité de notre trace NewItem pour visualiser les objets ramassés par chaque joueurs, et en faire des histogramme (Figure 7).

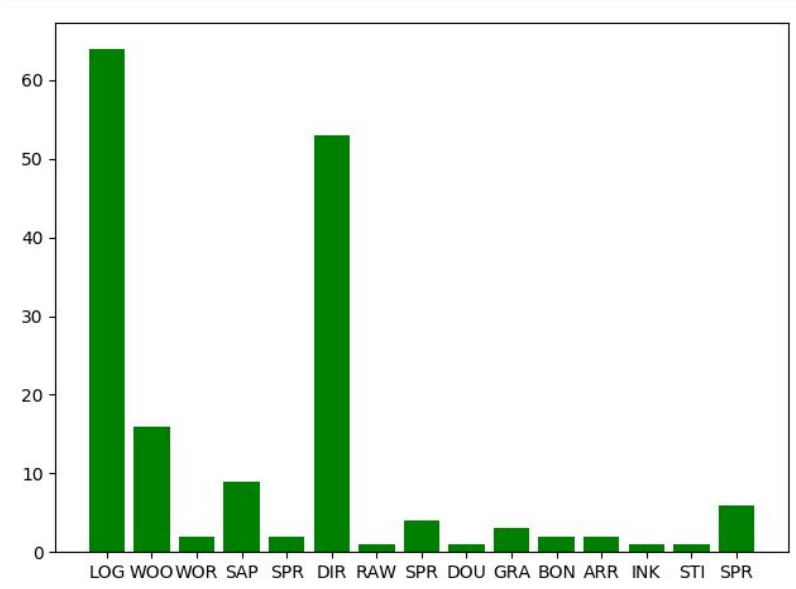

Figure 7 - *Exemples des WinItems pour Alice*

On observe qu'Alice a principalement obtenu des bûches et de la terre. On remarque aussi qu'elle a utilisé l'intégralité de ses bûches pour créer du bois (64 logs pour 16 bûches, et il faut 4 bûches pour faire 1 log). On observe entre autre qu'elle a obtenu 9 *spruce_fence* qui ont servi pour l'arche. On peut voir aussi les items plus rares qu'elle a obtenu, comme 1 *ink_sack* qu'elle a dû obtenir en tuant un poulpe, 1 *raw_fish* obtenu en tuant un poisson et 2 *bone + 2 arrows* obtenus en tuant 1 ou 2 squelettes.

Bob a obtenu entre autres 10 *raw_mutton* et 10 *wool* ce qui indique qu'il a tué entre 5 et 10 moutons. Il a aussi cueilli 30 roses qui ont servi pour le jardin de la maison, et il a principalement obtenus du sable et *1 sugar_cane* qui montrent qu'il a dû trouver un point d'eau.

Charlie a obtenu 1 *sulfur* en tuant un creeper, 1 *rotten_flesh* en tuant un zombie, et 2 *bone* en tuant un squelette, ce qui confirme nos hypothèses de combat de la partie précédente. Il a aussi chassé quelques moutons.

Dan a visiblement eu une partie très calme, il n'a obtenu que des blocs simples servant à la construction.

Finalement, on pourrait obtenir énormément d'informations en s'attardant plus longuement sur l'analyse de l'inventaire uniquement, et ça aurait pu être une approche intéressante et originale pour l'intégralité du projet.

III. Conclusion.

Les transformations permettent de modeler les traces comme on le souhaite et d'en tirer des informations intéressantes, nous avons cependant trouvé cela particulièrement laborieux sur le ktbs. Les méthodes built-in ont été difficiles à comprendre pour nous, et nos essais de fsa et de isparql pour les réécritures ont fini sur des abandons au profit de requêtes plus directes. La syntaxe a aussi été significativement ralentissante, chaque requête nous a nécessité de retourner lire le tutoriel, pour se rappeler les différents attributs attendus et comment les écrire. Nous avons le ressenti général que tout aurait été plus simple à faire directement en python, mais cela se justifie sûrement par notre manque d'expérience avec ktbs.

Nous avons réussi à comprendre globalement le déroulement de la partie à partir des traces fournies. La visualisation des constructions permet de se faire une bonne idée de l'état du monde. Les destructions permettent de se rendre compte de l'activité de chaque personnage. Les visualisations temporelles nous ont permis de retracer les événements importants dans le détail, comme la triste mort d'Alice. La tentative d'inventaire a tout de même apporté beaucoup d'informations et cela aurait été intéressant de creuser encore sur cette voie. Nous sommes globalement étonnés de la quantité d'informations que l'on a réussi à tirer de ces simples traces, sur une partie de seulement 30 minutes, en sachant que d'autres groupes ont trouvé encore d'autres informations.