Modifications:– Ajout du débuggage en annexe– Correction

Le Mans Université

Licence Informatique *2ème année*

Compte rendu de projet

Pac-Man

Taliesin Ambroise, Anthony Amiard, Hugo Bigan

Avril 2019

`https://github.com/anthonyamiard/pacman`

Table des matières

1 Introduction

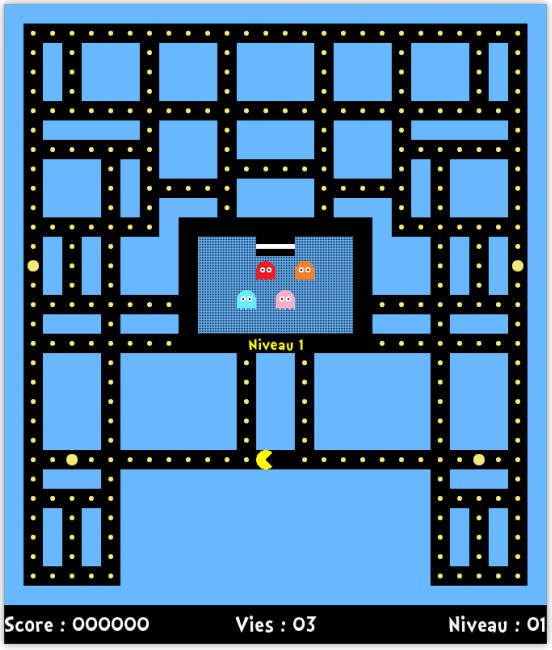
Pac-Man est un jeu vidéo édité par Namco en 1980. Le joueur doit déplacer le personnage Pac-Man dans un labyrinthe et manger toutes les pac-gommes (petites pièces jaunes présentes dans tout le labyrinthe) sans se faire attraper par les fantômes. Lorsqu'il mange une super pac-gomme (pac-gomme de plus gros diamètre), il peut manger les fantômes pendant une période de quelques secondes. Lorsque Pac-Man a mangé toutes les pac-gommes, il passe au niveau suivant.

Fonctionnalités implémentées

Dans notre projet nous avons implémenté les déplacements de Pac-Man, le rammassage des pac-gommes, la gestion du score, du nombre de vies et des niveaux, les déplacements des fantômes selon quatre comportements différents :

- un fantôme qui se dirige vers Pac-Man,
- un fantôme qui essaie de prendre Pac-Man en embuscade,
- un fantôme qui fuit Pac-Man,
- un fantôme qui se déplace aléatoirement.

Nous avons également implémenté la génération aléatoire du labyrinthe à chaque niveau, absente du jeu original.



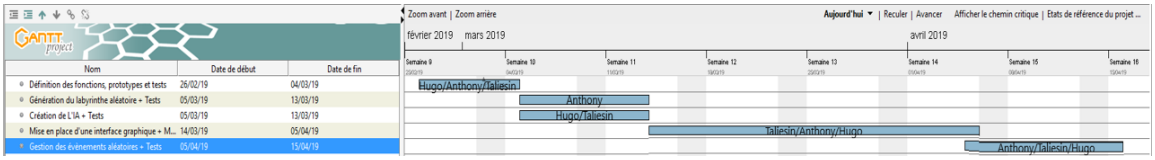


FIGURE 1 – Diagramme de Gantt

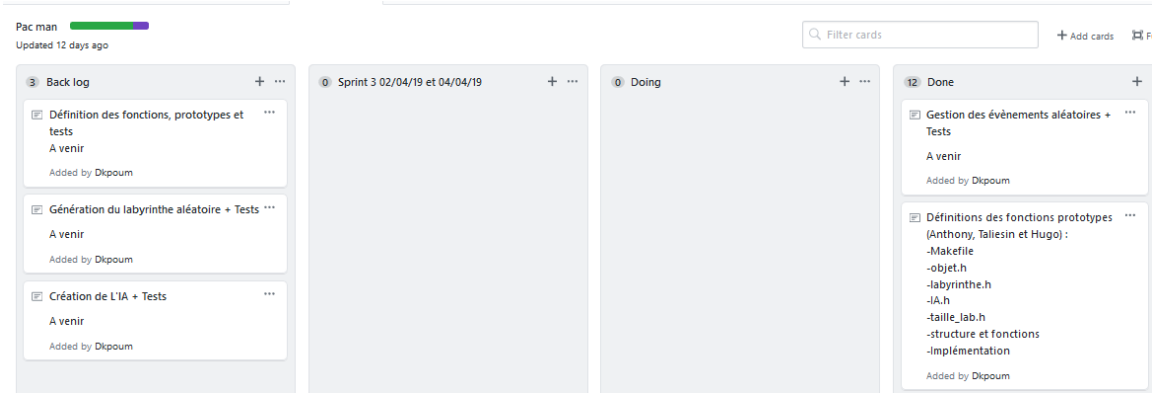


FIGURE 2 – Organisation en sprints

2 Organisation du travail

L'organisation du travail s'est mise en place au tout début du projet. Il a été décidé de créer des modules pour se partager le travail. Cette organisation s'est faite en groupe (modules, structure, fonctions principales) pour se mettre d'accord sur les bases à exploiter. Voir la documentation "Definiton_fonctions_prototype" associé.

Comme on peut le voir sur la figure 1 chaque module est géré par une personne (première énoncée) puis une autre pour la soutenir et l'aider dans celui-ci. Nous avons jugé nécessaire d'être minimum deux sur un module pour élargir les idées et les ajuster plus facilement entre nous.

De ce fait, des sprints se sont formés pour que le développement soit géré en temps et en heure.

Les tâches principales sont toujours visibles pour ne pas perdre le but du projet. Comme on peut le voir sur la figure 2 (dernier sprint), des tâches étaient placées à cette endroit et dédiées à certaines personnes. Grâce à ce procédé nous pouvons voir l'évolution du projet au niveau de chaque personne et l'aider si du retard est perceptible. Cette méthode clarifie le travail de groupe et nous permet de savoir ce qui est à faire et ce qui est fait.

3 Conception

Pour la conception de notre projet, nous avons commencé par gérer la création du labyrinthe qui ne doit pas comporter de sortie. Pour ce faire, dans le fichier `src/labyrinthe.c`, nous avons décidé de réaliser une matrice de caractères. Chacun de ces caractères correspond à une case du labyrinthe, pour savoir s'il s'agit d'un chemin, d'un mur, d'un chemin avec pac-gomme, etc.

Pour la conception de l'intelligence artificielle (IA), nous avons choisi de coder quatre comportements, et d'attribuer ces comportement à chacun des fantômes. Ces comportements ont été programmés dans le fichier `src/ia.c`. Le comportement le plus simple à coder a été la fonction qui consiste à faire prendre à un fantôme un chemin aléatoire en fonction de sa position. Le comportement le plus difficile à coder est la fonction pour faire prendre à un fantôme le chemin le plus court entre lui et Pac-Man. Enfin, un autre comportement fait fuir un fantôme de Pac-Man, et un autre pour que le fantôme anticipe les mouvements de Pac-Man et le prenne en embuscade. Dans tous les cas, chacune de ces fonctions retourne une valeur de type `coord_t` qui correspond ici à la coordonnée suivante au temps t+1 en fonction de celle de départ.

Les différents types d'objets (coordonnées, Pac-Man, fantôme, fruit), leur création et destruction en mémoire sont gérés dans le module `src/objets.c`.

Le module `src/score.c` et son fichier *header* correspondant `include/score.h` contiennent les définitions des nombres de points attribués à différentes actions (consommation d'une pac-gomme, d'une super pac-gomme, d'un fantôme ...) et une fonction permettant d'augmenter le score du joueur et de lui ajouter une vie s'il a atteint un certain palier.

Le module `gui/dessin.c` contient des fonctions automatisant le dessin d'image et de texte sur un *renderer* SDL, et une structure globale contenant la liste des sprites utilisés dans le jeu.

Le module `gui/gui_lab.c` contient les fonctions permettant d'afficher le labyrinthe en SDL et `gui/osd.c` contient les fonctions d'affichage du score, du nombre de vies et du niveau en bas de la fenêtre et de messages au milieu du labyrinthe lors de changement de niveau où à la fin du jeu.

Enfin le module `gui/mouv.c` contient les fonctions de gestion des déplacements des personnages à l'écran et de gestion des collisions.

4 Développement

Le développement du projet s'articule autour de quatre grandes phases comprenant un ou plusieurs modules : la génération du labyrinthe, l'intelligence artificielle des fantômes, la mise en place de l'interface graphique et la gestion des déplacements et collisions à l'écran.

4.1 Génération du labyrinthe

Le module `src/labyrinthe.c` contient les fonctions de génération aléatoire de labyrinthe. Le labyrinthe est représenté par une matrice de `N_LAB` par `M_LAB` [1] caractères, dont les différentes valeurs prises sont :

- `'c'` : chemin vide,
- `'p'` : chemin avec pac-gomme,
- `'s'` : chemin avec super pac-gomme,
- `'m'` : mur,
- `'b'` : boîte de départ des fantômes,
- `'e'` : entrée de la boîte des fantômes.

La fonction `genere_lab` prend en paramètre cette matrice et l'adresse d'une variable où écrire le nombre de pac-gommes présentes dans le labyrinthe. Elle va d'abord créer une matrice de même hauteur et moins large de moitié (le labyrinthe étant symétrique). Cette matrice est faite de murs, sauf dans les angles où l'on trouve des débuts de chemins et la boîte de départ des fantômes qui se situe au centre.

La fonction `chemin_alea` va ensuite être appelée aux points de départ des chemins. C'est une fonction récursive qui va tester s'il est possible de générer un chemin sur une case voisine choisie aléatoirement par `coord_alea`. Lorsque des coordonnées voisines valides ont été trouvées, la fonction va générer un chemin dans cette direction sur une distance de trois à douze cases, en appelant récursivement la fonction à certains endroits pour créer des ramifications (voir figure 3).

La validité des coordonnées est testée par la fonction `place_permise`. Elle est décomposée en trois niveaux :

- la place est permise (retourne 1),
- la place est interdite (retourne 0),
- la place est permise en dernier recourt (lors des phases de correction du labyrinthe, retourne 2).

Les règles pour qu'un couple de coordonnées soit valides sont les suivantes :

- les coordonnées doivent être comprises dans le demi-labyrinthe,
- la case ciblée n'est pas déjà un chemin,

1. définis dans `include/taille_lab.h`

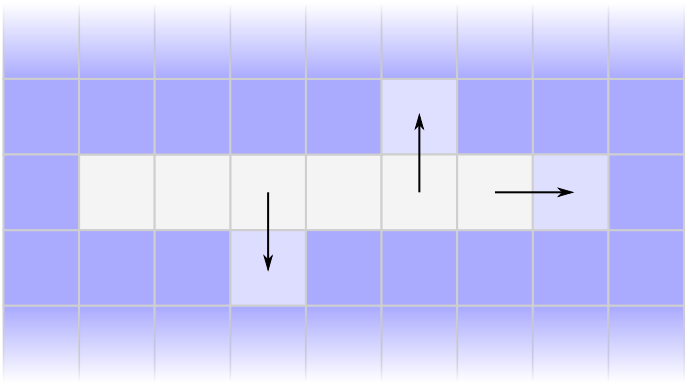


FIGURE 3 – Fonctionnement de `chemin_alea`
mur chemin

(a) Chemins de deux cases d'épaisseur (b) Chemins parallèles

FIGURE 4 – Fonctionnement de `place_permise`

— la case ciblée ne doit pas être entourée par trois chemins adjacents, afin d'empêcher la création d'un chemin de plusieurs cases d'épaisseurs (voir figure 4a),
— la case doit avoir au moins un chemin voisin,
— les murs ne peuvent pas faire une seule case d'épaisseur. Pour cela, la fonction recherche des chemins parallèles à celui ciblé par les coordonnées et interdit le placement si elle en trouve (voir figure 4b). Néanmoins, cette contrainte étant d'ordre esthétique (ne pas avoir de labyrinthe trop chargé), la place est autorisée en dernier recourt.

Le comptage du nombre de chemins voisins à une case est assuré par `nb_chemins_voisins` et `nb_chemins_voisins_demi` pour les demi-labyrinthes, et la fonction `est_chemin` permet de tester si une case est un chemin.

La génération des chemins crée des cul-de-sacs, pour les supprimer on utilise deux fonctions :

— `debouche_cds` qui, pour un couple de coordonnées donné, parcourt la matrice en ligne droite dans les quatre directions jusqu'à trouver un chemin vers lequel déboucher et crée des chemins entre les deux (voir figure 5). À ce stade-là, si `place_permise` retourne 2 (place permise en dernier recourt), la place sera considérée comme valide.

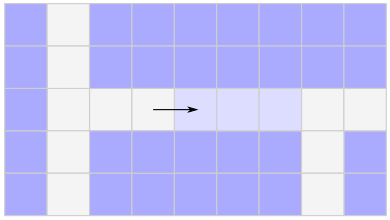


FIGURE 5 – Fonctionnement de `debouche_cds`

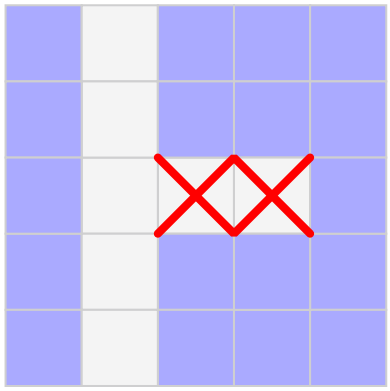


FIGURE 6 – Fonctionnement de `suppr_cds`

— `suppr_cds` qui, pour un couple de coordonnées donné, va remplacer le chemin par un mur et va s'appeler récursivement jusqu'au dernier carrefour (voir figure 6).

Chaque fonction est appelée dans une boucle qui parcourt toutes les cases du tableau.

Les jeux de tests de la génération de labyrinthe sont présents dans `test/test_lab.c` et portent sur les fonctions `coord_alea`, `est_chemin`, `nb_chemins_voisins`, `place_permise`, `debouche_cds` et `suppr_cds`.

4.2 Intelligence artificielle

Le module `src/IA.c` contient les fonctions de recherche de chemin des fantômes. Ils sont mis en place grâce aux coordonnées du labyrinthe en (x,y).

Les fonctions de chemin prennent toutes les mêmes paramètres : la matrice du labyrinthe, les coordonnées du fantôme et les coordonnées du joueur.

La fonction `chemin_alea` va simplement diriger le fantôme là ou il y a un chemin aléatoirement autour de lui grâce à `chemin_alea` défini juste avant.

La fonction `chemin_court` permet de trouver le chemin le plus court pour aller à la rencontre du joueur. On cherche d'abord le fantôme associé à la fonction pour ensuite chercher en-dessous de lui Pac-Man, sinon on remonte de la fin du labyrinthe jusqu'en haut pour le trouver et appliquer la méthode figure 7.

Cette méthode permet de vérifier toutes les coordonnées voisines et d'augmenter l'indice si l'on continue et si celui-ci est un chemin. Le retour va alors choisir la plus petite valeur autour de celle du joueur, retracer le chemin inverse, puis renvoyer les bonnes coordonnées (à coter du fantôme) pour rencontrer le joueur le plus rapidement possible. Si le chemin est trop éloigné alors la fonction `chemin_alea` prend le relais.

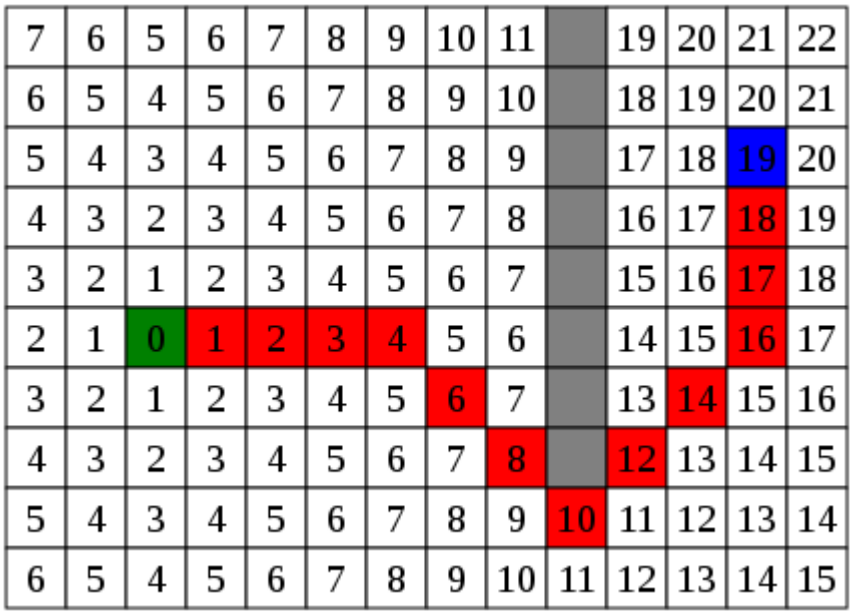


FIGURE 7 – Recherche de chemin

La fonction `chemin_anticipe` permet d'aller à la rencontre du joueur d'une autre manière. Si le joueur se déplace vers le haut alors le fantôme fera de même si c'est un chemin sinon il se reposera sur la fonction `chemin_court`. Cette fonction à pour principe de prendre le joueur en embuscade en renvoyant les coordonnées pour ce faire.

La fonction `chemin_fuir` va permettre de fuir le joueur en renvoyant les coordonnées pour que celui-ci lui échappe. C'est celle qui est utilisée par tous les fantômes en état de fuite, quand Pac-Man prend une super pac-gomme.

4.3 Interface graphique

L'interface graphique a été développée à l'aide de la bibliothèque SDL2. Le module `gui/dessin.c` contient des fonctions qui permettent d'automatiser le chargement et la conversion d'image et de texte en texture SDL et le chargement des *sprites* nécessaires au jeu dans une structure globale.

L'affichage du labyrinthe se fait grâce à la fonction `dessine_lab` du module `gui/gui_lab.c` qui parcourt toutes les cases du labyrinthe et appelle `case_img`, qui copie pour une case donnée la texture correspondante sur le *renderer* SDL.

La fonction `main` contenue dans le module `src/pacman.c` va, après avoir initialisé l'environnement graphique, parcourir une boucle dont une itération est une image. À chaque itération, les évènements sont testés, l'affichage du labyrinthe est mis à jour et les déplacements sont effectués. Ensuite une pause de quelques millisecondes est marquée pour avoir un nombre d'images

par seconde stable.

4.4 Gestion des déplacements

Les déplacements du joueur et des fantômes sont gérés par le module `gui/mouv.c`. Les fonctions du module sont appelées à chaque itération et permettent de mettre à jour la position du joueur et des fantômes sur le labyrinthe et de gérer les éventuelles collisions.

Dans cette partie, nous distingueront deux types de coordonnées différentes : les coordonnées d'une case sur le labyrinthe, que nous appelleront simplement coordonnées, et les coordonnées sur l'écran en pixels, que nous appelleront position. La position est égale à la coordonnée sur le labyrinthe multipliée par la taille d'une case en pixels (24 ici).

La fonction `deplace_coord` permet de changer la position d'un personnage dans une direction donnée, en testant si la destination voulue est bien un chemin.

Les fonctions `deplace_joueur` et `deplace_fantome` vont d'abord tester si le personnage se trouve entièrement sur une case (si le modulo de la position par la coordonnée est nul) pour mettre à jour ses coordonnées. Puis elles vont mettre à jour la position du personnage à l'aide de `deplace_coord` et enfin l'afficher sur le *renderer*.

La fonction `deplace_joueur` va également tester si le joueur se trouve sur une case contenant une pac-gomme pour la consommer : mettre à jour le labyrinthe pour la faire disparaître et augmenter son nombre de point. Idem s'il se trouve sur une case contenant une super pac-gomme, mais en plus de cela l'état des fantômes sera mis à jour en état de fuite.

La fonction `deplace_fantome` va, à chaque intersection ou face à un mur, chercher de nouvelles coordonnées vers lesquels se diriger. Si le fantôme est en état de poursuite du joueur, elle utilise la fonction de recherche de chemin du fantôme (dont le pointeur se trouve dans la structure) avec les coordonnées du joueur en paramètre. S'il est en état de fuite, alors elle utilise la fonction `chemin_fuir` avec les coordonnées du joueur en paramètre. S'il a été mangé par le joueur et qu'il doit retourner à son point de départ (état de retour), alors elle utilise la fonction `chemin_court` avec les coordonnées de l'entrée de la boîte en paramètre.

Lorsque les fantômes sont dans la boîte, leurs déplacement sont gérés directement par la fonction, sans recherche de chemin. S'ils sont en état d'attente (début d'un niveau, avant que le joueur n'appuie sur une touche), les fantômes se déplacent de haut en bas dans la boîte, s'ils sont en état de poursuite, ils se dirigent vers l'entrée horizontalement puis verticalement et s'ils sont en état de retour, ils se dirigent vers leur point de départ verticalement puis horizontalement.

Les collisions sont gérés grâce à deux fonctions :

— `collision` qui teste s'il y a collision entre le joueur et un fantôme,

— `gere_collision` qui prend en paramètre le joueur et les quatre fantômes et qui applique les traitement nécessaires en cas de collision.

La fonction `gere_collision` va tester pour les quatre fantômes si le joueur est en collision avec eux. Si tel est le cas, si le fantôme est en état de poursuite, alors le joueur perd une vie et tous les personnages reviennent à leur points de départ. S'il est en état de fuite, alors le joueur gagne le nombre de points associés au rang du fantôme consommé (s'il est le premier, le deuxième, le troisième ou le quatrième à l'être).

5 Résultat

5.1 Résultats attendus

Pour ce qui est des résultats attendus nous devions remplir les objectifs suivants :

— génération du labyrinthe aléatoire,
— mise en place d'une interface graphique SDL,
— intelligence artificielle pour chaque fantôme,
— jeu fluide et jouable.

5.2 Résultats obtenus

Tous les objectifs on été remplis suivant la demande initiale. Le jeu est jouable avec une diversité de labyrinthes générés. Les fantômes remplissent leur rôle, même si leur comportement est à affiner pour avoir une embuscade et une fuite plus « intelligente » et des fantômes mieux répartis sur l'ensemble du labyrinthe. Le joueur peut cumuler des points sur différents labyrinthes et mourir s'il ne possède plus de vies. Le fait de ramasser une super pac-gomme nous permet de « manger » les fantômes et ils reviennent à la chasse une fois la boîte rejointe.

6 Conclusion

Pour terminer, le projet pourrait être enrichi grâce à l'ajout des fonctionnalités suivantes :

- animation du *sprite* de Pac-Man,
- la vitesse des fantômes qui augmente selon le nombre de labyrinthes parcourus,
- affinage de certaines fonctions de recherche (`chemin_anticipe` et `chemin_fuir`),
- sauvegarde des scores (tableau des meilleurs scores),
- implémentation des bonus, fruits qui rapportent plus de points,
- implémentation d'un menu.

Ce projet est notre première expérience de travail informatique collaboratif avec les méthodes et outils associés (programmation modulaire, Git, diagramme Gantt, organisation en sprints ...) et nous a apporté de l'expérience sur la façon de travailler : se mettre d'accord ensemble au début du projet sur le découpage en module et prendre le temps de définir les structures de données et prototypes des fonctions de chaque module pour qu'ils puissent être utilisés par les autres pendant qu'elles sont implémentées.

En ce qui concerne l'organisation, il est parfois difficile de se concerter sur certains points même avec une bonne communication. Le travail collaboratif permet de gagner du temps en se partageant les tâches. Cependant la méthode de travail varie au sein du groupe, elle peut parfois poser problème au niveau des modules à relier entre eux ou pour comprendre le code de notre collègue.

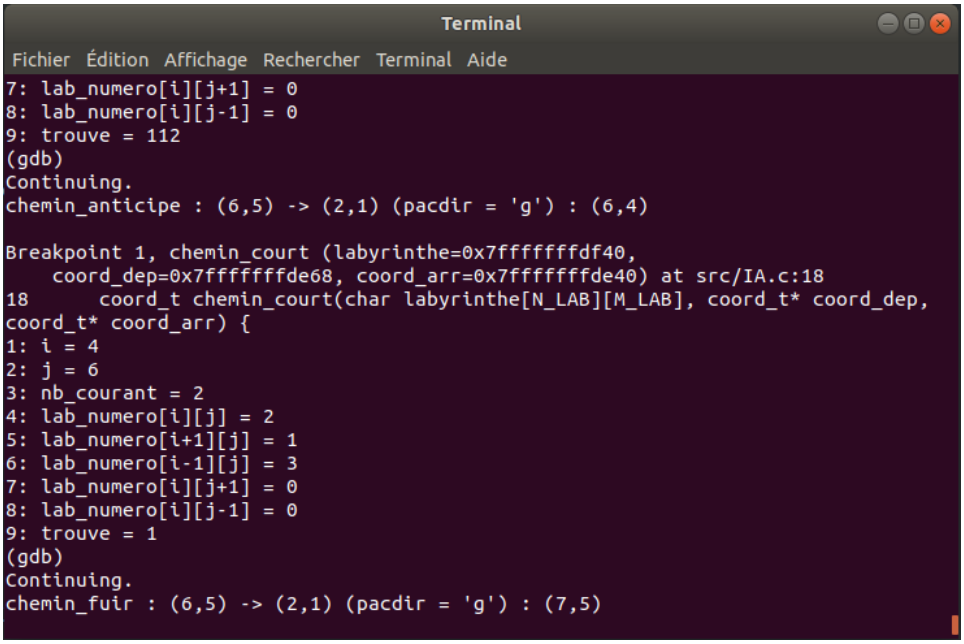


FIGURE 8 – Débuggage

7 Annexes

Pour ce qui est du débuggage nous avons dû utiliser GDB. Il a été utile surtout sur la partie où l'on a codé l'IA car nous ne trouvions pas l'erreur. Nous avons donc placé des *breaks* sur la fonction `chemin_court` car les résultats attendus grâce aux jeux de tests n'étaient pas obtenus. De ce fait, toutes les variables utiles on été taguées par la commande `display` et nous exécutions le programme pas à pas pour trouver l'erreur. Une fois l'erreur trouvée il nous fallait la résoudre mais celle-ci en entraînait d'autre. GDB nous a été très utile pour comprendre et résoudre les défauts de notre code. La figure 8 est une capture d'écran où l'on peut apercevoir le problème résolu.

Le problème était au niveau de `nb_courant` qui ne reprenait pas la bonne valeur et nous indiquait donc une mauvaise coordonnée en retour.

Explication : Si `lab_numero[i+7][j]` $->$ `nb_courant` de 7 on ne devait pas le reprendre 7 et avoir `lab_numero[i][j+1]` $->$ 8 mais `lab_numero[i][j+1]` $->$ 1.

D'autres problèmes moins gênants ont été résolus par ce procédé et l'étude de l'évolution des variables, comme l'exploration du `chemin_court` jusqu'à vingt cases.