Project CPA
BomberMan

Submitted by

Haotian Xue
Hejun Cao
M1 Science et Technologie du Logiciel 2023-2024
Sorbonne Universite (SU UPMC)

Under the guidance of

Binh-Minh Bui-Xuan
Laboratoire d'Informatique de Paris 6 (LIP6)
Centre National de la Recherche Scientifique (CNRS)
Sorbonne Universite (SU UPMC)

05/05/2024

Contents

1 Introduction

Ce projet vise à utiliser Java pour implémenter les fonctions de base du jeu classique BomberMan.

Dans le jeu, le personnage contrôlé par le joueur lâche une bombe qui attaquera l'ennemi avec une explosion en forme de croix après un délai. Dans le même temps, la bombe du personnage peut également détruire la boîte pour obtenir les accessoires à l'intérieur, qui peuvent être utilisés pour augmenter ou réduire la portée d'explosion de la bombe, ou pour restaurer sa propre santé.

Le jeu est principalement un mode de franchissement de niveau. Vous pouvez obtenir des points en battant des ennemis. Une fois que les points ont atteint le niveau cible, vous pouvez accéder au niveau suivant. Au fur et à mesure que le niveau progresse, les ennemis deviendront de plus en plus forts et les dégâts qu'ils causeront au joueur deviendront de plus en plus importants.

Le jeu implémente également un mode à deux joueurs. Dans ce mode, les joueurs doivent non seulement attaquer des monstres, mais également faire attention aux attaques d'un autre joueur. Ce mode consiste également à collecter suffisamment de points pour gagner, ou pour réduire la santé d'un autre joueur à zéro.

LACK OF technologies utilisées

LACK OF Gantt chart (IMAGE)

2 Architecture

L'architecture de ce projet est la suivante :

- **package go**

  Ce package est principalement utilisé pour définir divers objets dans le jeu, notamment des personnages contrôlés par le joueur, des monstres, des arbres, des murs, des buffs/debuffs, et bien plus encore.

  En même temps, il définit également les actions individuelles du joueur/monstre.

  Pour les joueurs, nous avons conçu son jugement de mouvement (s'il peut se déplacer dans la direction indiquée), la récupération des buffs/débuffs et la prévention des bombes.

  Pour les buffs et debuffs, nous avons mis en place une augmentation/diminution du nombre de bombes ou de la portée d'explosion.

- **package game**

  Ce package est principalement utilisé pour localiser les joueurs et les monstres, et calculer la prochaine position du joueur/monstre, ainsi que le niveau de difficulté actuel.

- **package engine**

  Il s'agit du package principal qui pilote l'implémentation du jeu. Il implémente principalement le cycle de vie de l'ensemble du jeu. Il est chargé de rafraîchir constamment tout l'écran de jeu, de charger et de mettre en œuvre l'animation (Sprite)

correspondant à chaque élément, le mouvement automatique des monstres et la prédiction d'action correspondante (où apparaîtra la prochaine image).

- **package launcher**

  Ce package est principalement utilisé pour définir des cartes par défaut et charger de nouvelles cartes.

- **package view/audio**

  Ces deux packages sont principalement utilisés pour charger des images et des sons.

3 Algorithme

3.1 Map Compression/Decompression

Notre carte est en fait éditée dans un format de tableau bidimensionnel, ce qui entraînera de très sérieux problèmes de stockage car les éléments de notre carte ne sont pas très complexes et comportent de nombreux objets en double.

Afin d'optimiser ce problème, nous avons conçu un algorithme de compression et de décompression pour les cartes afin de stocker et de transmettre les informations cartographiques plus efficacement.

La logique de la compression de carte est la suivante :

- *Step 1*

  Comme je l'ai dit précédemment, la carte est stockée sous la forme d'un tableau à deux dimensions. Nous parcourons donc les éléments de la carte ligne par ligne et obtenons le codage des caractères de chaque élément (via la fonction getCode), puis nous ajoutons tous ces codages à une chaîne de caractères.

- *Step 2*

  Chaque fois que nous finissons de parcourir une ligne de données, nous ajoutons un "x" pour indiquer une nouvelle ligne.

- *Step 3*

  Par la suite, nous utilisons un simple algorithme RLE (Run-Length Encoding) pour modifier les éléments répétés consécutifs sous forme de nombres + encodage. Par exemple, TTT (représentant trois arbres consécutifs) sera converti en 3T.

Pour la partie décompression, c'est en fait l'inverse complet du processus de compression. Tout d'abord, la chaîne après RLE est traitée et restaurée sous la forme où un seul objet occupe un caractère, puis, sur la base du caractère de nouvelle ligne "x", le tableau bidimensionnel correspondant à la carte est construit ligne par ligne.

3.2 IA pour Monstres

Dans n'importe quel jeu, les actions de l'IA de l'ennemi jouent un rôle très important, c'est pourquoi leurs algorithmes d'IA sont particulièrement importants.

Puisque la seule interaction entre les monstres et les joueurs dans notre jeu est le jugement de position, la partie la plus importante est en fait la logique de recherche de chemin du monstre, qui consiste à trouver le chemin le plus court vers l'emplacement actuel du joueur.

Nous avons d'abord conçu une plage de haine pour chaque monstre. Le mécanisme de recherche de chemin du monstre sera déclenché lorsque et seulement lorsque le joueur apparaîtra dans cette plage. À d'autres moments, le monstre se déplacera de manière aléatoire dans n'importe quelle direction.

Quant à la recherche active du monstre, nous avons choisi l'algorithme heuristique au lieu de l'algorithme A-star, Parce que notre carte est petite, l'avantage de vitesse de l'algorithme A-star n'est pas évident. Par rapport à la simplicité de l'algorithme de recherche heuristique, l'algorithme A-star est plus difficile à mettre en œuvre.

Notre conception heuristique est très simple :

- *Regle 1*

  Si le monstre est différent du joueur sur l'axe des x, déplacez l'axe des x du monstre pour qu'il se déplace vers le joueur.

- *Regle 2*

  S'il y a une différence sur l'axe y entre le monstre et le joueur, déplacez l'axe y du monstre pour qu'il se déplace vers le joueur.

- *Regle 3*

  Donnez la priorité au déplacement de l'axe x du monstre.

4 Gestion de Collisions

Gestion unifiée des erreurs : établissez un cadre global de gestion des exceptions pour capturer et répondre aux exceptions d'exécution dans le jeu.

Journalisation détaillée : implémentez un système de journalisation pour enregistrer les opérations des utilisateurs et les erreurs système afin de faciliter le débogage et l'analyse.

Nous utilisons des frameworks de journalisation tels que Log4j ou SLF4J pour mettre en œuvre des stratégies de journalisation détaillées. Ajoutez la journalisation aux points opérationnels clés, tels que la lecture et l'écriture de fichiers et les opérations utilisateur importantes.