RAPPORT (PI4) - PROJET DE PROGRAMMATION

BOMBERMAN

Image 1: Mode de jeu PVP

VYSHKA Tedi

THIRUKUMAR Kieran

RODRIGUEZ Lucas

MARTINEAU Clément

SOMMAIRE

1) OBJECTIFS DU PROJET
2) SCRUM, SPRINT & GIT (MÉTHODOLOGIE)
3) STRUCTURE DU PROGRAMME
4) ALGORITHMIQUE ET PROBLÈMES RENCONTRÉS
    a. Game Loop
    b. Bombes
    c. Déplacements
    d. Bonus
5) FONCTIONNALITÉS COMPLÉMENTAIRES
6) DIAGRAMME DE CLASSES

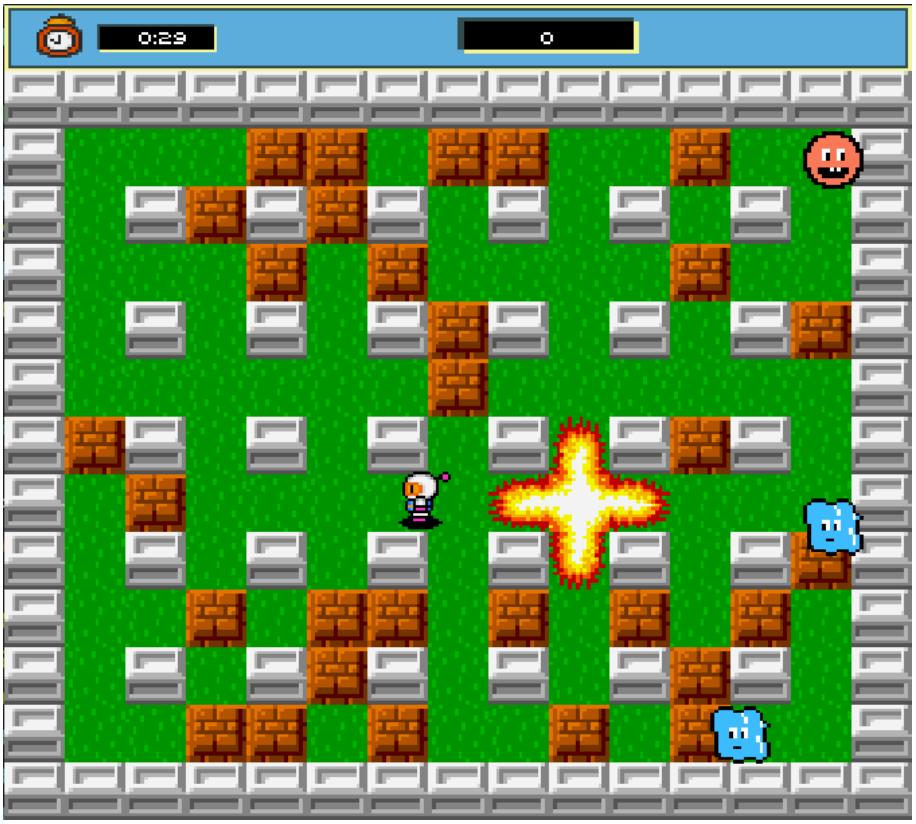


Image 2: Mode de jeu Monster

1. OBJECTIFS DU PROJET

Implémentation d'un jeu Bomberman avec deux modes de jeu:

- Joueur contre joueur (PVP)
- Joueur contre monstres (PvE avec friendly fire)

L'objectif du mode PVP est d'éliminer les autres joueurs et d'être le dernier en vie, tandis qu'une partie du mode PVE se termine quand le timer atteint 3 minutes ou si il n'y a plus aucun joueur en vie. Pendant le jeu les joueurs peuvent placer des bombes à retardement, casser certains murs (en marron) ou tuer les joueurs/monstres et récupérer des bonus qui sont cachés derrière les murs.

Les modes peuvent se jouer jusqu'à quatre joueurs.

2. SCRUM, SPRINT & GIT (MÉTHODOLOGIE)

Pour notre organisation, nous avons décidé de suivre la méthode Scrum en organisant des sprints chaque semaine. Ces sprints nous ont permis de connaître l'avancement de chacun sur les tâches à réaliser, corriger les problèmes ensemble et prévoir également les tâches à revisiter ou faire pour les semaines suivantes. De plus, nous avons utilisé GitLab et certains de ses outils.

3. STRUCTURE DU PROGRAMME

Le code est structuré avec le modèle de conception de logiciel Model-View-Controller (MVC) avec les trois paquets principaux:

- Le package Model contient tout ce qui concerne le jeu, le plateau, les instances des joueurs et des monstres.
- Les entrées saisies par les joueurs sont capturées par le package Controller, et après transmises vers le modèle afin d'effectuer les changements.
- Tout ça est affiché dans le GUI par le paquet View, qui crée les instances du GuiMenu, GuiBoard, GuiBar.

Une représentation graphique des paquets et classes est disponible dans le fichier "diagram.png", situé dans le dossier docs du projet.

4. ALGORITHMIQUE ET PROBLÈMES RENCONTRÉS

1. Game Loop

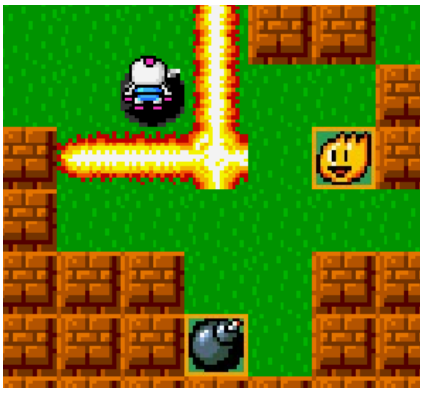
Notre jeu fonctionne grâce à une boucle principale. Le prototype de cette boucle fonctionnait de la manière suivante :

- une boucle `while` tournait en boucle en actualisant une variable avec le temps courant en milliseconde. A chaque fois que le temps courant dépassait la variable contenant le temps précédent + 1000/60 ms, alors on mettait à jour le Model et la Gui.

Cette implémentation posait un problème majeur. En effet, le `while` utilisait toute la puissance disponible pour tourner le plus vite possible. Cela faisait chauffer l'ordinateur et ralentissait le jeu.

Nous avons donc réfléchi à une solution différente qui permettrait de ne pas utiliser le CPU lorsqu'on attend d'avoir atteint 1000/60 ms. Nous avons opté pour la fonction de Java `Thread.sleep()` qui permet de mettre en attente le `Thread` pour une durée déterminée. Cela permet de réduire les performances au strict nécessaire.

2. Bombes



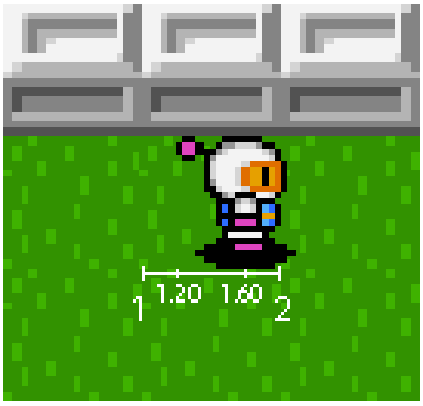
Lors de l'implémentation des explosions de bombes, nous avons rencontré un bug.

L'affichage du plateau de jeu se faisant de la gauche vers la droite et de haut en bas, les cases où étaient affichées les explosions étaient ré-affichées avec un simple bloc d'herbe (ou un bonus).

Pour corriger ce problème, nous avons ajouté les bombes sur le point d'exploser à une liste. Puis à la fin de l'affichage du plateau, nous faisons exploser chacune des bombes de cette liste.

3. Déplacements

Nous avons décidé de représenter le positionnement des joueurs et des bombes le plus simplement possible; notre modèle contient un tableau double d'objets Case comprenant les différents objets et variables le représentant, notamment Player et Bomb.



Le joueur dispose de deux variables (x,y) qui sont des coordonnées décimales, chaque unité représentant une case entière.

Lors du déplacement du joueur, la partie décimale varie selon la direction et la vitesse du joueur.

Cette implémentation nous permet donc de pouvoir dessiner précisément sur notre interface graphique la position des joueurs en multipliant leurs coordonnées par la largeur de la case en pixels.

4. Bonus

Pendant le jeu, les joueurs peuvent ramasser des bonus, cachés sous certains blocs cassables. Ces derniers permettent d'avancer plus vite, ou d'ajouter diverses améliorations sur les bombes.

Les bonus implémentés sont les suivants:

- Bonus qui augmente la portée de la bombe d'un bloc. 
- Bonus qui augmente la portée de deux blocs. 
- Bonus qui permet de placer une bombe supplémentaire. 
- Bonus qui permet de casser plusieurs blocs d'un coup. 
- Bonus cumulable qui rend le joueur plus rapide. 
- Bonus qui permet au joueur de pousser les bombes. 

5. IA et BOT

Initialement, nous pensions faire deux types d'IA : un premier type dit, "naïf", qui reproduit les mouvements basiques et les actions primaires des monstres que l'on peut trouver dans les jeux Bomberman, le déplacement de ces IA aurait été généré aléatoirement.

Un deuxième type, dit "intelligent", qui reproduit le comportement d'un joueur. Cette IA aurait dû pouvoir jouer et prendre des décisions selon les contraintes de l'environnement du jeu (ennemis aux alentours, possibilité de récupérer des bonus avoisinant, s'écarter d'une bombe)[1] et agir en conséquence.

La première méthode reposait sur un algorithme de pathfinding effectuant un parcours en largeur. Notre problème fut que les déplacements n'étaient pas implémentés en case par case mais avec des nombres décimaux.

L'adaptation de l'algorithme avec notre implémentation des déplacements étant trop complexe, nous avons opté pour une nouvelle version.

Nous vérifions si le bot parvient à quitter sa case et suivre le chemin correctement, avant qu'il ne change de direction ou de type d'action. Cela ne fonctionnant pas non plus, nous avons décidé de négliger la réalisation de l'IA intelligente et de réaliser seulement des IA naïves pour les monstres.

6. Restart button

L'implémentation d'un bouton pour revenir au menu s'est avérée surprenament compliquée. Suite à de nombreuses méthodes et tentatives infructueuses, nous avons opté pour une solution avant tout fonctionnelle.

Lors de la fin du jeu, nous affichons les boutons "quit" et "restart", lorsque l'on appuie sur ce dernier, le jeu est arrêté de la même manière que pour une "pause", toutes les références aux objets utilisés sont supprimées, un nouveau menu est lancé, qui lui même créera un nouveau modèle. Cette méthode permet de faire un redémarrage sans laisser une boucle tournant en arrière plan.

[1] Documentation sur l'implémentation d'une IA de bomberman: *Beating Bomberman with Artificial Intelligence(2018)*

5. FONCTIONNALITÉS COMPLÉMENTAIRES

En addition des fonctionnalités du jeu de base, nous avons pensé à intégrer un nouveau mode de jeu, le Joueurs contre Monstres (PvE). Comme indiqué précédemment, l'objectif de ce mode est d'éliminer les monstres qui apparaissent sur la carte, et cela dans un temps imparti. Nous avons repris des caractéristiques générales et simples des monstres, en créant deux types de monstres :

- Les monstres de couleur bleue ou `FlyingMonster`, sont des monstres qui volent et traversent les murs cassables. Ils se déplacent par rangée de lignes ou colonnes et à la détection d'un joueur, décide d'aller dans sa direction pour l'éliminer.
- Les monstres de couleur marron ou `WalkingMonster`, se déplacent aléatoirement et plus rapidement.

Comme évoqué dans la partie susnommée, ces monstres sont implémentés d'une manière "naïve". Les monstres apparaîtront en quantité proportionnelle au nombre de joueurs dans la partie, chaque monstre tué est remplacé par un autre et apparaît sur le plateau là où les joueurs ne sont pas présents.

Nous avons aussi ajouté un menu de pause activé lors d'un appui sur la touche "ECHAP". Initialement notre programme pour être le plus simple possible n'était composé d'un seul `Thread` général, celui créé lors de l'exécution du programme. Vouloir implémenter cette pause nous a fait réaliser qu'un autre `Thread` serait nécessaire nous permettant de mettre en pause ainsi seulement une partie du programme, la boucle du jeu. Pour réaliser cette pause, un appel à la fonction `wait()` est effectué puis `notifyAll()` pour reprendre la partie.

6. DIAGRAMME DE CLASSES