Bomberman clone game using Pygame

Levchenko Maksym
ČVUT FIT
levchmak@fit.cvut.cz
2024-01-06

Introduction

The game is an engaging and dynamic adaptation of the classic Bomberman concept. Players navigate through mazes strategically planting bombs to destroy obstacles and blow up the enemies. The game features a diverse range of elements, including a randomized map generation, both multiplayer and single-player modes, enemy AI with distinct algorithms(Dijkstra and DFS), and different kinds of power-ups available to the players. The player's objective is to strategically deploy bombs to eliminate enemies and destroy obstacles.
Used algorithms

Depth-First Search (DFS)

1. The DFS algorithm is employed in the **`dfs`** method of the **`Enemy`** class.

2. It explores possible paths, considering safe and unsafe tiles, and backtracks if needed.

3. The exploration is limited based on the bomb limit, and planting a bomb is considered during the search.

4. The algorithm also shuffles the movement directions randomly for exploration.

Dijkstra's Algorithm:

1. The Dijkstra's algorithm is employed in the **`dijkstra`** method of the **`Enemy`** class.

2. It uses a grid of `Node` objects representing the game map.

3. The algorithm considers safe and unsafe tiles, bomb positions, and enemy positions to find the shortest path.

4. The movement directions are shuffled randomly for exploration.

Problems in development

The major problem with player's movement was a requirement of an extremely high precision for positioning a player's model to fit between two blocks. To solve this complication, I have come up with a smoothening technique, which provides a much more likable experience for the user. The **solution** is straightforward: In the `move` method of the **`Player`** class, if the player's position is not perfectly aligned with the tiles and the movement is only in one dimension, it adjusts the position to align with the tiles.

Another problem was an inability of artificial intelligence to detect if an initially built path was afterwards blocked by another enemy’s position or a player. Also it didn’t take into account possible explosions, which can appear at the path. **Solution** is also very simple: In the **`move`** method of the **`Enemy`** class, if the enemy reaches the center of the next tile, it checks if the path is blocked. If a blockage is detected, the **`movement_path`** and **`path`** lists are cleared, and a new path is calculated.

Moreover, there was a complication with running tests due to problems with relative imports. To solve this inconvenience, I had to create a whole copy of **src** directory with relative imports and named it **src_test,** and only then it ended up working.

Conclusion

I believe that I have successfully developed a user-friendly game that includes a variety of convenient features, as well as an intelligent AI system that can adapt to changes in the game environment. Moreover, I have left room for further expansion of the game, such as the addition of power-ups to enhance the overall gaming experience, and the inclusion of various game modes to keep players engaged.

References

1. "Bomb Party - The Complete Set," OpenGameArt. [Online] Available: https://opengameart.org/content/bomb-party-the-complete-set

2. Dijkstra's Algorithm, Wikipedia. [Online] Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

3. Depth-First Search, Wikipedia. [Online] Available: https://en.wikipedia.org/wiki/Depth-first_search

4. "Animations of Sprite in Pygame," ClearCode. [Online] Available: https://www.youtube.com/watch?v=MYaxPa_eZS0&ab_channel=ClearCode