Departamento de Informática da Faculdade de

Ciências da Universidade de Lisboa

Inteligência Artifical em Jogos

2022/2023

Final Project

Super Mario Bros

Student	Number
José Siopa	fc60716
Martim Costa	fc60504

June 25, 2023

1 Introduction

The gaming industry has seen huge strides in artificial intelligence (AI), which has improved user experiences and produced more captivating games. Our goal in this project is to create an AI system that has been especially designed for the well-known Super Mario video game. The goal of our project is to create an intelligent agent capable of exploring Super Mario's world and making deft judgments by utilizing the information and materials we have learned in class.

This report aims to detail the work done for the creation of the AI agent for the Super Mario game.

2 Objectives

- **Level completion**: The main goal of our AI system is to offer the AI agent the ability to effectively explore and finish a specific level in the Super Mario game. To accomplish this, he must navigate around a number of barriers, including platforms, gaps, and enemies. Our AI agent should try to finish the level quickly by using intelligent decision-making and adaptable techniques.
- **Collecting Coins**: In addition to finishing the level, our AI agent should be able to gather the coins that he finds possible to obtain around the level. This would add another component other than just finishing the level.
- **Eliminating Goombas**: Another important goal is to give the AI agent the ability to deal with the game's enemies, particularly the Goombas. Goombas are frequent enemies that the player must avoid or eliminate. In order to ensure the agent's survival and progression through the level, our AI agent should contain techniques for avoiding or eliminating these enemies.

We want to demonstrate the capability of our AI agent in terms of completing levels, gathering coins, and dealing with in-game enemies by attaining these goals.

3 Approach

Firstly, we decided to make use of a pathfinding algorithm, so our agent could traverse the level in order to get to our previously defined goals in a more efficient way. For this we used the A* (A star) pathfinding algorithm, since we know where the objectives are, it makes for a fast path calculation with this algorithm.

After that, we decided that we would use a Behavioral Tree for the different actions Mario could take (these ended up being either following the path given or jump over goombas) as well as for the conditions that trigger these actions.

4 Implementation

4.1 Search algorithm

For the implementation of the A* pathfinding algorithm, we used a more simplified map version of the real one, only comprised of one pixel with a different color for each type of enemy/block/object (ex: goomba, normal block, flag), this way we reduce the time needed to calculate by simplifying the map in a reasonable way. These simplified maps are stored in the game's files and are not built during runtime.

The method starts by creating a list of open nodes, where the first node is the starting position with a *g_cost* of 0 and an *f_cost* of the heuristic cost to the target position. The heuristic cost is calculated using the heuristic function, which calculates the Manhattan distance between the current position and the target position. The method then enters a loop where it repeatedly selects the node in the open set with the lowest *f_cost* and removes it from the open set. If the selected node is the target position, the method reconstructs the path from the target position to the starting position by tracing back through the parents of each node in the path. If the selected node is not the target position, the method generates successors for the node by calling the *generate_successors* function, which returns a list of tuples representing the possible moves from the current position. The method then iterates over each successor and checks if it is already in the closed set. If it is not, the method calculates the new *g_cost* for the successor by adding the cost of the current move to the *g_cost* of the current node. If the successor is not in the *g_scores* dictionary or if the new *g_cost* is lower than the current *g_cost*, the successor is added to the open set with a new *f_cost* calculated using the heuristic function. The method continues this loop until it finds the path to the target position or until it runs out of open nodes to evaluate. If the method does not find a path to the target position, it returns None.

We then take this list and use the designated actions for the correct transformed coordinates in the real map/game.

4.2 Behaviors

For the actions we have a Behavioral Tree that depending on the action given by the A* pathfinding algorithm or the distance to the enemy will decide on what action to take. It will do the left action, right action, jump right action and jump left action if Mario is in the correct location equivalent to the nodes. It will also make the jump actions in case of a goomba being in a certain threshold that differs if they're to the left or the right of Mario.

In case a goomba is detected, the action to avoid/eliminate takes priority over following the path. If Mario deviates too much from the intended path, another path will be created from the current Mario's location.

For the jump actions we defined how many nodes Mario can jump from his current position, and the option to jump is possible if one of the available nodes he can jump into has a walkable block directly under that node. We also verify if Mario can drop from a higher position to a lower position if there is a block that he would be able to land on and if it is a walkable block. We also had to consider that Mario couldn't jump immediately when he touched the correct block since many times it would result in the jump missing its expected target.

For the objectives we did not add the enemies to the pathfinding algorithm, since the simplified map doesn't update it would cause problems to our agent by it trying to hit a goomba that could not be there at that point in time.

5 Results

Our AI agent is able to complete the first level effortlessly, gathering all the coins while avoiding and killing most goombas, as well as avoiding the pitfalls. It is necessary to remember that the goombas are not part of our objectives, therefore making the action to jump over them is only reactionary and only occurring if they are within the threshold implemented. It was able to accomplish the basic of finishing the level and gathering the coins that we set out to do so we considered this a success.

However, for the next level, the agent could not find a path to the coins for yet to know reasons. For example, Mario did not want to cooperate only defaulting to our jump right option that uses if it has no other actions or paths to take. We feel that could be something in the original code that could be causing conflicts with our pathfinding but as of writing this report we couldn't find out why.

Another problem to solve would be including moving platforms in the search algorithm. By our experiments, repeatedly calculating a path every frame causes major performance issues, so there's a problem we could not also solve.

Since the Super Mario world has so many world variables to consider for the AI, including timings, enemies, the inertia of the movement, moving platforms, etc, there is no easy solution for a modular AI capable of completing these worlds effortlessly.

6 Future Work

Scale our agent to be able to work with every Mario level, besides fixing the part of not working in other levels, we also need to consider the different blocks, like moving platforms and such special blocks. We would also want to add the enemies to the goals to be able to accurately eliminate them, this would require updating the path search algorithm every frame with their positions. This would require threading to delegate the tasks and get better performance.

Demonstration link: https://www.youtube.com/watch?v=SqZcPbr3HUk