deti
departamento de eletrónica,
telecomunicações e informática

Development of an autonomous agent for the DigDug game

Artificial Intelligence, 2023/2024

Diana Miranda 107457
Rúben Garrido 107927

Defense strategies

Run away from fire

Fygars can throw fire, which immediately kills DigDug. Every frame, our algorithm checks for Fygars close to DigDug and, upon taking a decision, evaluates if any of them can spill fire to it.

The `will_enemy_fire_at_digdug` function takes the new DigDug position and the enemy and checks **whether the enemy position will be dangerous or not** for DigDug.

The function contains a lambda expression that tests several position conditions, based on the enemy direction. Those generally include the 4 consecutive **positions to where the enemy is looking to.**

The number 4 was chosen to give DigDug a margin of one pixel away from the worst-case scenario.

Avoid dangerous positions

Before making any final decision about the next position DigDug will move to, it is checked **whether any of the enemies in the game can reach the location DigDug is interested in.**

For each `check_dist_all_enemies` call, the new direction is calculated based on the new position, and then a few possible scenarios are tested.

If any condition fails, it will check another direction. This process repeats until a safe one to advance is found.

Even though this lowers DigDug's attack effectiveness, this strategy focus on every enemy attack possibility, so that there is little to no chance for DigDug to die.

Defense strategies

Rocks and map boundaries

Rocks can be useful to kill enemies, but DigDug can die because of them.

Before moving, the new position is checked against rocks and their positions, so that they won't kill DigDug.

Also, the map boundaries are checked so that a move won't take a non-expected result.

For example, going east when DigDug is on the very right side of map produces no effect. **Without this approach, the move would be either neutral or negative**, as the DigDug would not move and one of the enemies could attack it.

Helper function with fallback

To reduce boilerplate code and make the agent less bug prone, a helper function `dig_map` was created, which returns a key to the server.

It takes the desired direction and a list of fallback directions in case the desired one does not meet the required conditions. If the fallback list is also empty, it simply returns an empty string (i.e., DigDug stays in the same place).

Those conditions include the ones already mentioned (enemies, fire, map boundaries, and rocks).

Attack strategies

Defense over attack

Before attacking an enemy, we guarantee that:

- No other enemy can kill DigDug, either by position moves or through fire;
- The map is dug in the chosen enemy direction

If all conditions are met, the code returns “A” (i.e., the attack key). Otherwise, it moves to a defensive position, with fallback options as well.

We chose this strategy since it would make no sense getting killed while attacking, since that would mean one less life.

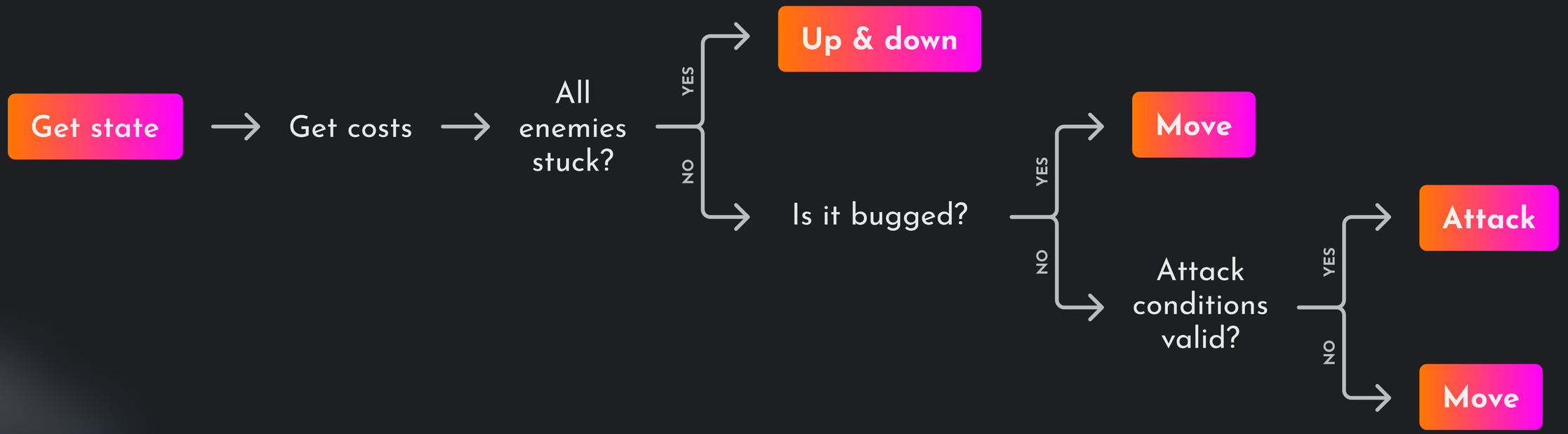
A* algorithm

The choice of which enemy DigDug will attack is determined by the implementation of the A* algorithm, which always seeks the enemy with the lowest cost in relation to DigDug, resulting in the closest enemy.

Initially, we considered using A* to check if there was an already digged path for any enemy or to determine which enemy would require less dig by DigDug. The goal was to avoid unnecessary digging, restricting the available paths for enemies.

However, due to challenges in realizing this idea and since we had already started implementing the tree search, we chose to continue using that approach.

Architecture diagram



Benchmarking

Although our code doesn't have any Python 3.11 or 3.12 exclusive features, we decided to test and benchmark each Python version due to internal CPython improvements in 3.10. We tested them by measuring the amount of time to get a key string when a new state arrives from the server, 10000 times per version.

Since the differences are very minimal and thus not taken into account, we can conclude that **all three tested versions have the same performance** in this context.