CI213 Assessment 2 – AI for Games

# Description of task:

The task is to program a game that represents an adventurer travelling though unknown territory, in this case named a dungeon. Both the adventurer and monsters they encounter have 3 main attributes: Intelligence, Speed and Weapon, with monsters having an additional trait relating to their personality/behaviour. The adventurer also has 3 items they can use only once, to temporarily increase one of their stats to win a fight or escape. The goal is to create a program that keeps the adventurer alive for as long as possible. Given the monsters and their abilities are randomly generated every round, the adventurer will face some monsters which they cannot defeat, thus the adventurer will survive more rounds in some dungeons, and less in others depending on the spawns of the monsters.

# How it works:

The program first begins with asking whether the player would like to start. If yes, then the main game loop (start()) is called. The first thing this function does is set up the variables used in the game, primarily:

* The player is set to true (alive)
* UI counters are initialised
* The player’s stats and inventory are created
* The monster’s variable is created

I use various logs to the console to show the player what they are starting off with, then the main loop starts.

The first job of the game loop is to reset the player stats and item Booleans (used to control when an item has been used) so that the effects of items are removed when a new round begins. Then the spawn() function is called to create a new monster. The spawn() function uses ThreadLocalRandom to generate 4 random numbers within certain parameters, which are then assigned and returned in an array to signify the monster’s Intelligence, Speed, Weapon and Weapon type, and Behaviour. This information is printed to the console, as well as the differences between the player’s stats and the monster’s. These differences are used by the AI during decision making.

Now the main decision loop runs. First, we check if the monster is scared, and if so, do nothing to avoid a fight. If it isn’t scared, the first check is done to see if we can win at hypnotising the enemy. If not, the second check is to see if we can run away. There is no check to see if we can definitively win using a weapon as the player stat is 2, and the minimum enemy stat is 1, therefore we can only win with a 50% chance or with a tie-breaker. The next best outcome is to find a tiebreaker in intelligence or weapon skill as the enemy will run away and the adventurer will survive. If there are no tie-breakers, we check if there is a 50/50 chance of winning either an intelligence or weapon battle (this could however kill the player, hence why is comes near the bottom of the decision tree). As a last resort, if there are no 50/50 chances, we check to see if using an item (starting with the player’s weakest ability (weapon) so that the best is saved for last) will help. Finally, if all else failed, we do nothing and see what happens.

For the decision loop I decided to use a while loop so that I could easily break out of it once a decision was reached. This could have also been achieved using a switch case break system.

After the decision has been chosen by the AI, we handle the decision. This is where most feedback is printed to the console. Each decision handler is similar, each first checks if an item was used, then checks to see if the player would win or lose the battle. In the case of the weapon and intelligence decisions, there is also a chance the player’s skill will be 1 more than the monster’s, resulting in a 50% chance battle. Here a number is generated between 0 and 100, and if the result is greater than or equal to 50, the player succeeds, otherwise the player is defeated. In the case of the player doing nothing, the result will depend on the monster’s personality type. If they are scared the monster will run away, if they are curious you they will watch as you pass by, however if they are aggressive, they will attack you with a weapon attack.

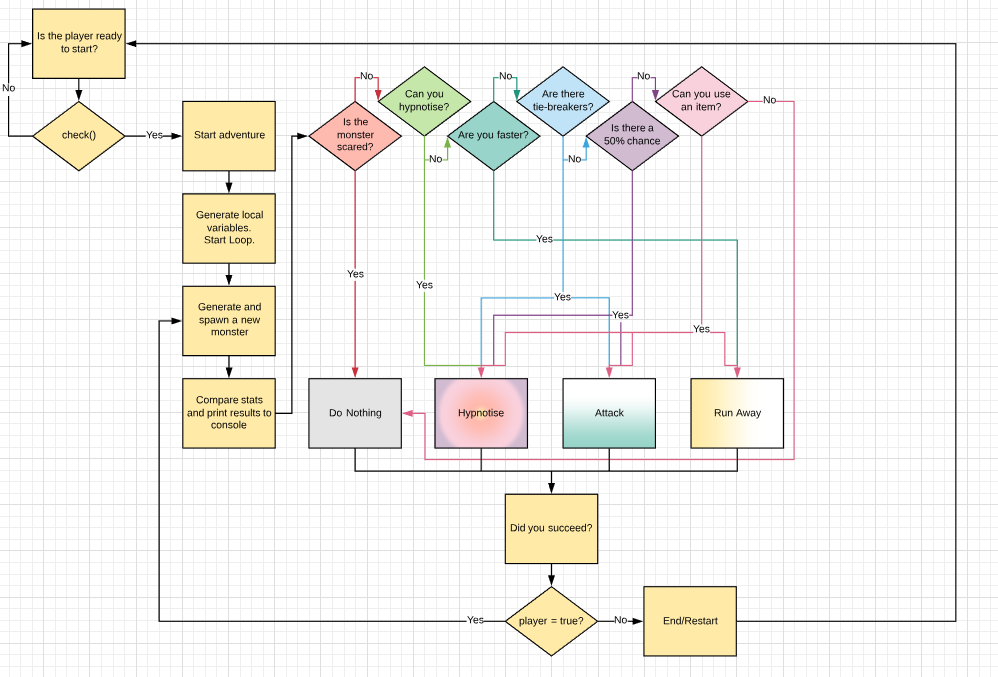
Once the loop is broken (the player has died), a final scoresheet is printed to the console so show how many rounds were survived, and how many monsters the player killed, hypnotised or managed to escape from. If you would like to run another dungeon after this, you can type “start” again to see a different result.

Figure : A visualisation of the game loop

A video demonstration has been provided. As the original video surpassed the 3 minute mark, it was slightly sped up to be under the 3 minute mark.

# Test Cases:

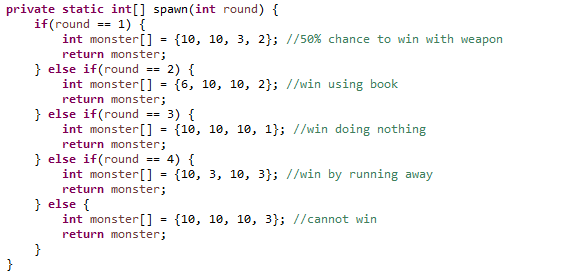
For the test cases, I made 5 example monsters which show how the decision tree works, showing what it’s outcome should be. These cases were chosen as they show most of the different decisions the AI will make depending on the scenario. Here monster stats were raised beyond that of 5 for clarity of the affecting stat. The test cases are in the spawn(int round) function, called instead of the spawn() function. This means only 5 rounds will be played, however the first example monster will lead to a 50/50 battle between the player and the monster so the result of the first round will differ depending on the number generated. You can re-run dungeons by typing start in the console once they have finished. The code and outputs in the console will look like this.

Figure : TestCase.java spawn function



Figure : TestCase output, Round 1 loss

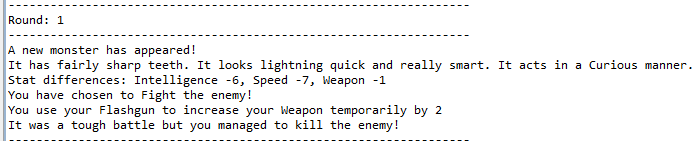


Figure : TestCase output, Round 1 win