### Level 1 – Behavior Trees

For the first level we were tasked with the implementation of behavior tree for the Orcs that included the Patrol task. For this purpose, we added the option on the *LevelCreator* script to include patrol points for the characters (represented by the letter *p*). This made it possible to generate patrol points for different maps without having to add empty objects directly on the editor.

Since we had implemented the *Selector* composite task, we derived the *OrcPatrolTree* from that class. As the guide pointed out, it was impossible, at first, to interrupt the *Patrol* task when the player was spotted. To counter this issue, we changed the order of the tasks within the *OrcPatrolTree.* This solved the problem, given that the *Selector* runs the first successful task. By putting the *Patrol* task after *Pursue* task, the Orcs started behaving correctly.

Uma imagem com texto

Descrição gerada automaticamenteFor the shout task (class *HearOrcShout*), we added an audio source to the Orcs for the auditive part and a canvas with text in world space for the visual effect. If the Orcs hear a shout they’ll move to the position where the shout came from. If the Orc stops shouting (either because it stopped seeing the player or was killed), the Orcs will resume their “normal” behavior. This led to the problem where the Orcs would pursue the player but not attack it, since the *LightAttack* task was ordered after all other actions. As such, it was placed first on the list and modified to take into account whether the target was in range or not, which solved the problem.

Figure 1. Visual representation of the Pursue task with shout

### Level 2 – GOB and GOAP

For the GOB algorithm, the *Rest* action was a challenge, since the player would continuously choose to rest to recover HP instead of achieving the end goal (even with the Sleeping NPCs option enabled). This happened because the Survive goal would achieve bigger numbers than the Be Quick goal. To counter this, and to prevent the player from never healing, we limited the value of the Be Quick goal to half of the player’s maximum HP. With this, the AI would now heal until the relative safety of over half HP and then continue to play, not being stuck resting until the full health was recovered.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Time Until Win (Average) | Best Discontentment (Average) | Total Actions Taken (Average) | Win Rate (Best of 3) |
| GOB (sleeping NPCs) | 118 s | 11,08 | 5 | 100% |
| GOB (stochastic world) | 182 s\* | 21,01\* | 20\* | 50% |
| GOB (non-stochastic) | 134 s | 21,01 | 13 | 100% |
| GOAP (sleeping NPCs) | 156 s | 0.96 | 12 | 100% |
| GOAP (stochastic world) |  |  | 14 | 50% |
| GOB (non-stochastic) |  |  |  | No |

\*Statistics only apply in case of victory

### Level 3 – Sir Uthgard’s Actions

### Level 4 – MCTS

Figure 2. Difference between optimal and chosen paths

Note: Sometimes the player will run out of time because the *NavMesh* will not choose the shortest path. For example, in figure 2 the optimal path from X to the chest is displayed in green and the path chosen by Unity is displayed in red. To account for this situation, we changed duration calculation to a more pessimistic approach.

### Secret Level 1 – Optimizing World State Representation

### Secret Level 2 – Limited Playout MCTS

### Secret Level 3 – Comparison of MCTS variants

### Secret Level 4 – Additional Optimization